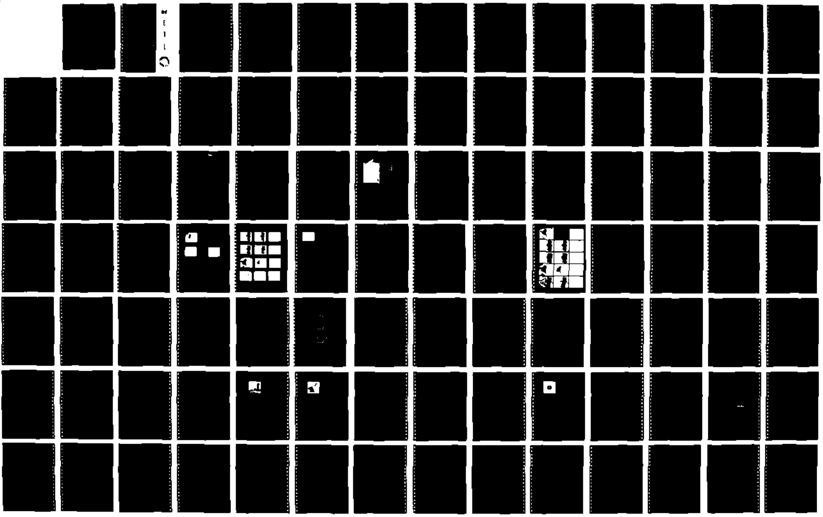
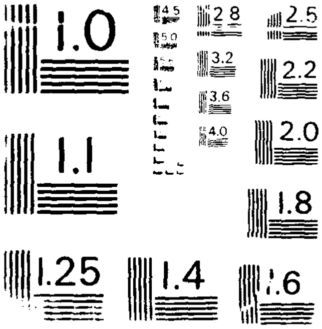
AD-A212 489 PARALLEL ALGORITHMS FOR COMPUTER VISION(U)
MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL
INTELLIGENCE LAB T POGGIO JAN 89 ETL-0529
UNCLASSIFIED DACA76-85-C-0010 F/G 12/1 NL




1.0
4.5
5.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

2

Parallel Algorithms for Computer Vision, Third Year Report

Tomaso Poggio

Massachusetts Institute of Technology
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Massachusetts 02139

January 1989

Approved for public release; distribution is unlimited.

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

U.S. Army Corps of Engineers
Engineer Topographic Laboratories
Fort Belvoir, Virginia 22060-5546

Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release;
Distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ETL-0529

6a. NAME OF PERFORMING ORGANIZATION
Massachusetts Institute
of Technology

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
U.S. Army Engineer
Topographic Laboratories

6c. ADDRESS *(City, State, and ZIP Code)*
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

7b. ADDRESS *(City, State, and ZIP Code)*
Fort Belvoir, VA 22060-5546

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Defense Advanced Research
Projects Agency

8b. OFFICE SYMBOL *(If applicable)*
ISTO

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DACA76-85-C-0010

8c. ADDRESS *(City, State, and ZIP Code)*
1400 Wilson Boulevard
Arlington, VA 22209-2308

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Parallel Algorithms for Computer Vision - Third Year Report

12. PERSONAL AUTHOR(S)
Tomaso Poggio

13a. TYPE OF REPORT
Annual

13b. TIME COVERED
FROM 8/31/87 TO 8/31/88

14. DATE OF REPORT *(Year, Month, Day)*
1989 January

15. PAGE COUNT
153

16. SUPPLEMENTARY NOTATION Previous reports in series by T. Poggio and J. Little:
ETL-0456 Parallel Algorithms for Computer Vision January 1987 (covers 8/85 to 8/86)
ETL-0495 Parallel Algorithms for Computer Vision Second Year Report March 1988 (8/86-8/87)

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Computer vision
Parallel algorithms and architectures

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This is the third annual report for Contract DACA76-85-C-0010, entitled "Parallel Algorithms for Computer Vision - Task B," sponsored by the Defense Advanced Research Projects Agency (DARPA), and administered by the U.S. Army Engineer Topographic Laboratories (ETL).

The time period covered is the second year that we have had the Connection Machine (CM) available to us. During the same period of time, we successfully demonstrated the Vision Machine system processing images and recognizing objects through the integration of several visual cues. The first version of the Vision Machine system, which is based on the CM and uses an Eye-Head robot as an input device, is now complete and functional. In parallel with the development of the Vision Machine, we have also continued to study the performance of alternative, nonconventional architectures for navigation. The body of this report gives an overview of the results of our research during the third year of funding. Details can be found in the appendices.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
George Lukes

22b. TELEPHONE *(Include Area Code)*
(202) 355-2700

22c. OFFICE SYMBOL
CEETL-RI

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

PREFACE

This report describes work performed under contract DACA76-85-C-0010 for the U.S. Army Engineer Topographic Laboratories, Fort Belvoir, Virginia 22060 by Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The Contracting Officer's Representative was Mr. George Lukes.

Parallel Algorithms for Computer Vision – Third Annual Report

Contract Number DACA76-85-C0010
Defense Advanced Research Projects Agency
U.S. Army Topographic Laboratories

Tomaso Poggio

August 31, 1987 – August 31, 1988

Third Year Report

TABLE OF CONTENTS

1. SUMMARY

1.1 An Overview

This is the third annual report for Contract DACA76-85-C-0010, entitled "Parallel Algorithms for Computer Vision - Task B." sponsored by the Defense Advanced Research Projects Agency (DARPA), and administered by the U.S. Army Engineer Topographic Laboratories (ETL).

The time period covered by this report is the second year that we had the Connection Machine (CM) available to us. During the same period of time, we successfully demonstrated the Vision Machine system processing images and recognizing objects through the integration of several visual cues. The first version of the Vision Machine system, which is based on the CM and uses an Eye-Head robot as an input device, is now complete and functional. In parellel with the development of the Vision Machine, we have also continued to study the performance of alternative, nonconventional architectures for navigation. The body of this report gives an overview of the results of our research during the third year of funding. Details can be found in the appended publications.

1.2 The Vision Machine

The Vision Machine is a computer system that integrates several vision cues to achieve high performance in unstructured environments for the tasks of recognition and navigation. It is also a test-bed for our theoretical progress in low- and high-level vision algorithms, their parallel implementation, and their integration. As discussed in previous reports, the Vision Machine consists of a movable two camera Eye-Head system, the input device, and a Connection Machine, the main computational engine. During the third year of funding, we developed and implemented several parallel early vision algorithms which compute edge detection, stereo, motion, texture, and surface color in close to real-time. We have now integrated these algorithms with an integration stage based on the technique of coupled Markov Random Field (MRF) models that provides a cartoon-like map of the discontinuities in the scene. In recent months we have also obtained a partial labeling of brightness edges in terms of their physical origin. As planned, we have interfaced the output of our integration stage with a model-based parallel recognition algorithm. We are beginning a project together with Electrical Engineering faculty (with non-DARPA funding) to develop analog and hybrid Very Large Scale Integration (VLSI) implementations of the main Vision Machine components.

2. ACHIEVEMENTS IN THE THIRD YEAR

Below is a brief overview of our main achievements.

2.1 The Vision Machine

The overall organization of the Vision Machine system is based on parallel processing of the images by independent algorithms or modules corresponding to different visual cues. Edges are extracted using Canny's edge detector. The stereo module computes disparity from the left and right images. The motion module estimates an approximation to the optical flow from pairs of images in a time sequence. The texture module computes texture attributes (such as density and orientation of textons. The color algorithm provides an estimate of the spectral albedo of the surfaces, independently of the *effective illumination*, that is, illumination gradients and shading effects, as suggested by Hurlbert and Poggio (see Poggio, 1985).

The measurements provided by the early vision modules are typically noisy, and possibly sparse (for stereo and motion). They are smoothed and made dense by exploiting known constraints within each process (for example, that disparity is smooth). This is the stage of *approximation* and *restoration* of data, performed using a Markov Random Field (MRF) model. Simultaneously, discontinuities are found in each cue. Prior knowledge of the behavior of discontinuities is exploited, for instance the fact that they are continuous lines, not isolated points. Detection of discontinuities is aided by the information provided by brightness edges. Thus each cue, disparity, optical flow, texture, and color, is coupled to the edges in brightness.

The full scheme involves finding the various types of physical discontinuities in the surfaces, *depth discontinuities* (extremal edges and blades), *orientation discontinuities*, *specular edges*, *albedo edges* (or marks), and *shadow edges*, and coupling them with each other and back to the discontinuities in the visual cues. So far we have implemented only the coupling of brightness edges to each of the cues provided by the early algorithm. As we will discuss later, the technique we used to approximate, to simultaneously detect discontinuities, and to couple the different processes, is based on MRF models. The output of the system is a set of labeled discontinuities of the surfaces around the viewer. In our implemented version of the system, we find discontinuities in disparity, motion, texture, and color. These discontinuities, taken together, represent a "cartoon" of the original scene, which can be used for recognition and navigation (together with interpolated depth, motion, texture, and color fields, if needed).

2.2 Hardware

2.2.1 The Eye-Head System

Because of the variety of visual information processed by the Vision Machine, a general purpose image input device is required. This device is the Eye-Head system.

The Eye-Head system consists of two CCD cameras ("eyes") mounted on a variable-attitude platform ("head"). The apparatus allows the cameras to be moved as a *unit, analogous* to head movement. It also allows the lines of sight of the cameras to be pointed independently, analogous to eye movement. Each camera is equipped with a motorized zoom lens ($F1.4$, focal length from 12.5 to 75mm), allowing control of the iris, focus, and focal length by the host computer (currently a Symbolics 3600 Lisp Machine). Other hardware allows for repeatable calibration of the entire apparatus.

Because of the size and weight of the motorized lenses, it would be impractical to achieve eye movement by pointing the camera lens assemblies directly. Instead, each assembly is mounted rigidly on the head, with eye movement achieved indirectly. In front of each camera lens is a pair of front surface mirrors, each of which can be pivoted by a galvanometer also mounted rigidly on the head. The mirrors are positioned to provide two degrees of freedom in aiming the cameras. At the expense of a more complicated imaging geometry, this allows for a simpler and faster control system for the eyes.

The head is attached to its mount via a spherical joint, allowing head rotation about two orthogonal axes (pan and tilt). Each axis is driven by a stepper motor coupled to the drive shaft through a harmonic drive. The latter provides a large gear ratio in conjunction with very little mechanical backlash. Under control of the stepper motors, the head can be panned 180 degrees from left to right and tilted 90 degrees (from vertical-down to horizontal). Each of the stepper motors is provided with an *optical shaft encoder* for shaft position feedback (a closed-loop control scheme is employed for the stepper motors). The shaft encoders also provide an index pulse (one per revolution) which is used for joint calibration in conjunction with mechanical limit switches. The latter also protect the head from damage due to excessive travel.

The overall control system for the Eye-Head system is distributed over a micro-processor network (UNET) developed at the MIT AI Lab for the control of vision robotics hardware. The UNET is a "multi-drop" network supporting up to 32 microcomputers under the control of a single host. The micros normally function as network slaves, with the host acting as the master.

In this mode, the micros only "speak when spoken to," responding to various network operations either by receiving information (command or otherwise) or by transmitting information (such as status or results). Associated with each micro on the UNET is a local 16-bit bus (UBUS), which is totally under the control of the micro. Peripheral devices such as motor drivers, galvanometer drivers, and pulse width modulators (PWMs), to name a few, can be interfaced at this level.

At present, two micro-processors are installed on the Eye-Head UNET: one for the galvanometer, and one for both the motorized lenses and the stepper motors. The processors currently employed are based on the Intel 8051. Each of these micros has an assortment of UBUS peripherals under its control. By making these peripherals sufficiently powerful, each micro's control task can remain simple and manageable. Code for the micros, written in both assembly language and C, is facilitated by a Lisp-based debugging environment.

2.2.2 Our Computational Engine: The Connection Machine

The Connection Machine (CM) is a powerful fine-grained parallel machine which has proven useful for implementation of vision algorithms. The machine is described in the appendices (*Introduction to Data Level Parallelism. Thinking Machines Technical Report 86.14*). We now have a 16K CM1 and a 8K CM2 with floating point hardware. Many vision problems must be solved by a combination of communication modes on the Connection Machine. The design of these algorithms takes advantage of the underlying architecture of the machine in novel ways. There are several common elementary operations used in this discussion of parallel algorithms: routing operations, scanning, and distance doubling.

Routing

Memory in the Connection Machine is associated with processors. Local memory can be accessed rapidly. Memory of processors nearby in the NEWS network can be accessed by passing it through the processors on the path between the source and the destination. At present, NEWS accesses in the machine are made in the same direction for all processors. The *router* on the Connection Machine provides parallel reads and writes among processor memory at arbitrary distances and with arbitrary patterns. It uses a packet-switched message routing scheme to direct messages along the hypercube connections to their destinations. This powerful communication mode can be used to reconfigure completely, in one parallel write operation taking one router cycle, a field of information in the machine. The Connection Machine supplies instructions so that many processors can read from or write to the same location, but since these memory references can cause significant delay, we will usually only consider exclusive read and exclusive write instructions. We will usually not allow more than one processor to access the memory of another processor at one time. The Connection Machine can combine messages at a destination by various operations, such as logical AND, inclusive OR, summation, and maximum or minimum.

Scanning

The *scan* operations (Blelloch, 1987) can be used to simplify and speed up many algorithms. They directly take advantage of the hypercube connections underlying the router, and can be used to distribute values among the processors and to aggregate values using associative operators. Formally, the *scan* operation takes a binary associative operator $\div$, with identity 0, and an ordered set $[a_0, a_1, \ldots, a_{n-1}]$, and returns the set $[a_0, (a_0 \div a_1), \ldots, (a_0 \div a_1 \div \ldots \div a_{n-1})]$. This operation is sometimes referred to as the *data independent prefix operation* (Kruskal et.al., 1985). Binary associative *operators include minimum, maximum*, and *plus*.

The four scan operations *plus-scan*, *max-scan*, *min-scan*, and *copy-scan* are implemented in microcode, and take about the same amount of time as a routing cycle. The *copy-scan* operation takes a value at the first processor and distributes it to the other processors. These scan operations can take *segment bits* that divide the processor ordering into segments. The beginning of each segment is marked by a processor whose segment bit is set, and the scan operations start over again at the beginning of each segment.

The *scan* operations also work using the NEWS addressing scheme, termed *grid-scans*. These compute the sum and quickly find the maximum, copy, or number values along rows or columns of the NEWS grid. For example, for each pixel *grid-scans* can be used to find the sum of a square region with width $2m + 1$ centered at the pixel. This sum is computed by the following steps. First, a *plus-scan* accumulates partial sums for all pixels along the rows. Each pixel then gets the result of the scan from the processor m in front of it and m behind it; the difference of these two values represents the sum, for each pixel, of its neighborhood along the row. We now execute the same calculation on the columns, resulting in the sum, for each pixel, of the elements in its square. The whole process only requires a few *scans* and routing operations, and runs in time independent of the size of m. The summation operations are generally useful to accumulate local support in many of our algorithms, such as stereo and motion.

Distance Doubling

Another important primitive operation is *distance doubling* (Wyllie, 1979; Lim, 1986), which can be used to compute the effect of any binary, associative operation, as in *scan*, on processors linked in a list or a ring. For example, using *max*, *distance doubling* can find the extremum of a field contained in the processors. Using message-passing on the router, *distance doubling* can propagate the extreme value to all processors in the ring of N processors in $O(\log N)$ steps. Each step involves two *send* operations. Typically, the value to be maximized is chosen to be the hypercube-address. At termination, each processor in the ring knows the label of the maximum processor in the ring, hereafter termed the *principal processor*. This labels all connected processors uniquely, and nominates a processor as the representative for the entire set of connected processors. At the same time, the distance from the *principal procesor* can be computed in each processor. Figure 4 shows the propagation of values in a ring of eight processors. Each processor initially, at step 0, has the address of the next processor in the ring, and a value which is to be maximized. At the termination of the i^{th} step, a processor knows the addresses of processors $2^i - 1$ away and the maximum of all values within 2^{i-1} processors away. In the example, the maximum value has been propagated to all eight processors in $\log 8 = 3$ steps.

2.3 Early Vision Algorithms and their Parallel Implementation

We have described the early vision algorithms and their implementation in a previous report (see appendices, *The MIT Vision Machine*, Proceedings of the Image Understanding Workshop, 1988). Although we have done substantial work to improve some of these algorithms (most notably stereo), we will not describe this work here; details can be found in the appended papers. We will describe, however, two new algorithms for color and texture.

2.3.1 Color

The color algorithm that we have implemented is a very preliminary version of a module that should find the boundaries in the surface spectral reflectance function, that is, discontinuities in the surface color. The algorithm relies on the idea of *effective illumination* and on the *single source* assumption, both introduced by Hurlbert and Poggio (see Poggio et.al., 1985).

The single source assumption states that the illumination may be separated into two components, one dependent only on wavelength and one dependent only on spatial coordinates, and generally holds for illumination from a single light source. It allows us to write the image irradiance equation for a Lambertian world as

$$I^\nu = k^\nu E(x,y)\rho^\nu(x,y) \tag{5}$$

where I^ν is the image irradiance in the νth spectral channel ($\nu = red, green, blue$), $\rho^\nu(x,y)$ is the surface spectral reflectance (or albedo), and $E(x,y)$ is the effective illumination which absorbs the spatial variations of the illumination and the shading due to the 3D shape of surfaces (k^ν is a constant for each channel and depends only on the luminant). A simple segmentation algorithm is then obtained by considering the equation

$$H(x,y) = \frac{I^r}{I^r + I^g} = \frac{k^r\rho^r}{k^r\rho^r + k^g\rho^g} \tag{6}$$

which changes only when ρ^r or ρ^g or both change. Thus H, which is piecewise constant, has discontinuities that mark changes in the surface albedo independently of changes in the effective illumination.

The quantity $H(x,y)$ is defined almost everywhere, but is typically noisy. To counter the effect of noise, we exploit the prior information that H should be piecewise constant with discontinuities that are themselves continuous, non-intersecting lines. As we will discuss later, this restoration step is done using a MRF model. This algorithm works only under the restrictive assumption that specular reflections can be neglected. Hurlbert [1988] discusses in more detail the scheme outlined here and how it can be extended to more general conditions.

2.3.2 Texture

The texture algorithm is a greatly simplified parallel version of the texture algorithm developed by Voorhees and Poggio [1987]. Texture is a scalar measure computed by summation of texton densities over small regions surrounding every point. Discontinuities in this measure can correspond to occlusion boundaries, or to orientation discontinuities, which cause foreshortening. Textons are computed in the image by simple approximation to the methods presented in Voorhees and Poggio [1987]. For this example, the textons are restricted to blob-like regions without regard to orientation selection.

To compute textons, the image is first filtered by a Laplacian of Gaussian filter at several different scales. The smallest scale selects the textural elements. The Laplacian of Gaussian image is then thresholded at a non-zero value to find the regions which comprise the blobs identified by the textons. The result is a binary image with non-zero values only in the areas of the blobs. A simple summation counts the density of blobs, the portion of the summation region covered by blobs, in a small area surrounding each point. This operation effectively measures the density of blobs at the fine scale, while also counting the presence of blobs caused by large occlusion edges at the boundaries of textured regions. Contrast boundaries appear as blobs in the Laplacian of Gaussian image. To remove their effect, we use the Laplacian of Gaussian image at a slightly coarser scale. Blobs caused by the texture at the fine scale do not appear at this coarser scale, while the contrast boundaries, as well as all other blobs at coarser scales, remain. This coarse blob image filters the fine blobs: blobs at the coarser scale are removed from the fine scale image. Then summation, whether with a simple scan operation or Gaussian filtering, can determine the blob density at the fine scale only. This is one example in which multiple spatial scales are used in the present implementation of the Vision Machine.

2.4 The Integration Stage and MRF

Whereas it is reasonable to assume that combining the evidence provided by multiple cues, for example, edge detection, stereo, and color, should provide a more reliable map of the surfaces than any single cue alone, it is not obvious how this integration can be accomplished. The various physical processes that contribute to image formation, *surface depth, surface orientation, albedo* (Lambertian and specular component), illumination, are coupled to the image data, and therefore to each other, through the imaging equation. The coupling is, however, difficult to exploit in a robust manner since it depends critically on the reflectance and imaging models. We argue that the coupling of the image data to the surface and illumination properties is of a more qualitative and robust sort at locations in which image brightness changes sharply and surface properties are discontinuous, in short, at edges. The intuitive reason for this is that at discontinuities, the coupling between different physical processes and the image data is robust and qualitative. For example, a depth discontinuity usually originates a brightness edge in the image, and a motion boundary often corresponds to a depth discontinuity (and a brightness edge) in the image. This view suggests restoring the data provided by early modules or the following integration scheme. The results provided by stereo, motion, and other visual cues are typically noisy and sparse. We can improve them by exploiting the fact that they should be smooth, or even piecewise constant (as in the case of the albedo) between discontinuities. We can exploit *a priori* information about generic properties of the discontinuities themselves, for instance, that they are usually continuous and non-intersecting.

The idea, then, is to detect discontinuities in each cue, such as depth, simultaneously with the approximation of the depth data. The detection of discontinuities is aided by information on the presence and type of discontinuities in the surfaces and surface properties (see Figure 1), which are coupled to the brightness edges in the image.

Notice that reliable detection of discontinuities is critical for a vision system, since discontinuities are often the most important locations in a scene: depth discontinuities, for example, normally correspond to the boundaries of an object or an object part. The idea is thus to couple different cues through their discontinuities and to use information from several cues simultaneously to help refine the initial estimation of discontinuities, which are typically noisy and sparse.

How can this be done? We have chosen to use the machinery of Markov Random Fields (MRF), initially suggested for image processing by [Geman and Geman 1984]. This technique and our integration scheme are described in detail in the appended papers.

A few disclaimers are in order at this point. We have chosen to use MRF models because of their generality and theoretical attractiveness. This does not imply that stochastic algorithms must be used. For example, in the cases in which the MRF model reduces to standard regularization [Marroquin et.al., 1987] and the data are given on a regular grid, the MRF formulation leads not only to a purely deterministic algorithm, but also to a convolution filter.

We are now beginning to define deterministic algorithms that are either equivalent to a MRF formulation or are a good approximation to the stochastic Monte Carlo algorithms. More specifically, we expect that the probabilistic formulation of a MRF is in a sense too general, and therefore too inefficient. Remember that MRF models are quite general; for example, regularization can be regarded from a probabilistic point of view as an instance of a MRF.

2.5 The Recognition Stage

The *output of the integration stage* provides a set of edges labeled in terms of physical discontinuities of the surface properties. These are a good input to model-based recognition algorithms. We have interfaced the integration stage of the Vision Machine as implemented so far with the Cass algorithm. We have used only discontinuities for recognition; we plan to also use the information provided by the MRFs about the surface properties *between* discontinuities.

2.6 Results

We have used the Vision Machine to compute a "cartoon" of visible scenes containing a variety of objects such as model planes and office scenes. This cartoon of discontinuities has been used to recognize objects of which models were available. The whole system works in parallel on the Connection Machine. Examples of the ouput of the integration stage are shown in the appended papers. We have not yet published our results on recognition.

2.7 A VLSI Vision Machine

Our Vision Machine consists mostly of specialized software running on a general purpose computer, the Connection Machine. This is a good system for the present stage of experimentation and development. Now that we have perfected and tested the algorithms and the overall system, it makes sense to compile the software in silicon in order to produce a faster, cheaper, and smaller Vision Machine. We are presently starting a project with Electrical Engineering faculty to use analog and digital VLSI technology to develop some initial chips as a first step toward this goal.

3. OTHER RECOGNITION WORK ON THE CONNECTION MACHINE

Willie Lim has implemented an object model aquisition and recognition scheme using the Connection Machine to do parallel library search. The domain was the "rocks world," a world of rigid naturally occurring objects which are not easily described using conventional techniques. The obvious application for such a vision system is landmark recognition in outdoor navigation.

The key idea in this work was to model irregulariy shaped objects as a qualitative silhouette graph. Silhouette features invariant to small changes in viewpoint, such as points of maximum curvature, are used to parse the silhouettes into chunks, which are then described qualitatively. During model aquisition, the system directed a mobile camera (mounted on the end of a robot manipulator) to take new images from views with maximum uncertainty on the frontier of the partially constructed model. The resulting models are organized on a qualitative Guassian sphere. During matching of an unknown object, all previously built models are matched simultaneously on the Connection Machine. Each node in each model graph is assigned a unique processor. Each node tries to match itself against features in the input description and propagates matching constraints to its neighbors in its own model graph. Thus matching happens in bounded time regardless of the size of the library, up to the capacity of the CM.

4. OTHER ARCHITECTURES

Most of our work has focused on the Connection Machine, as we had originally proposed, in order to establish the strength and utility of fine-grained parallel architectures for vision within a navigation task. Thanks to the in-house work of Rodney Brooks on the Mobile Robot, we have found it interesting to contrast the fine-grained architecture of the CM with the subsumption architecture used by Brooks.

4. 1 Architectures for Robot Navigation

Previously, we had begun work using the subsumption architecture for navigation, in which perception was carried out by other processes. During the last year, we also applied subsumption directly to building a visual perception system for a mobile robot.

The subsumption architecture is a naturally parallel system which uses message passing between simple computational elements to localize decisions to precisely the relevant aspects of the world. We extended this architecture by adding visual pathways where image data could be sent between the computational elements. The computational elements were augmented to include array processing primitives.

Two implementations of this architecture were built. One used 8-bit microprocessors to do real-time object recognition in depth images. The outputs were used to direct a robot with an onboard arm to locate and retrieve known objects in a dynamically changing cluttered environment. The other implementation simulated a parallel machine on a Lisp Machine, and did real-time processing of five frames per second from a standard black and white camera, enabling the robot to follow corridors and locate and follow slow moving objects. The first test used a pipeline of simple 8-bit microprocessors (6800s) with a cycle time of onel microsecond to achieve real-time vision in depth images.

The complete system including a laser light striper, a parallel processor, and a manipulator mounted on a mobile robot. The light striper uses a standard black and white CCD camera with an appropriate filter. Odd interlaces were discarded to allow for stabilization of the mechanical system, while each even interlace, every one-thirtieth of a second, provided a horizontal scanline of disparity. In this way, every 1.067 seconds a 32-high by 256-wide disparity image was collected. As each scanline was collected, it was expanded to a kilobyte by the addition of three temporary bytes to each data byte. The image was represented everywhere by these expanded scanlines. As each scanline was collected, it was piped into the first in a series of microprocessors. Each microprocessor had 8K of EPROM, and 128 bytes of scratch RAM onboard. A 2-Kbyte RAM associated with each processor held the two most recent scanlines of the image. The processors were connected in a tree, rooted at the light striper, and every 33.3 milliseconds a scanline of data was shifted through the system in a 0.5 millisecond burst. This left each processor with time for about thirty instructions per pixel of the original depth image.

The recognition problem is partitioned into a number of simple operations on the image, each of which was mapped to a processor. Early in the tree of processors, the operations were generic (e.g. depth-based segmentation), but towards the leaves of the tree, the operations were quite specific (e.g., shape matching for a specific shape). The implemented version used a tree of processors six deep, giving a one-fifth second latency between completion of the collection of the image and the parallel output from a set of leaf nodes of object identifications and localizations. The system has been used to reliably locate and grasp target objects in cluttered dynamically changing environments.

In the second test of this approach, a single black and white CCD camera was mounted on a mobile robot base. The images were subsampled down to 32 by 28, and a new image was taken every 0.2 seconds.

A reliable system for approaching and following moving objects was built by first building a simple but unreliable system, then adding a second simple and also potentially unreliable system, to make an overall reliable system. Once this initial reliable system had been built, it was easy to add on top of it a few more computational modules and have it do a quite different task: drive along corridors.

The simplest system compared successive images using pixel differences an binary thresholding to detect areas of motion. The program servoed the robot base so that the centroid of this motion area was lined up in a particular spot in the images. Servoing the base in this way causes the robot to chase the detected motion. This system is very unreliable due to noise in the images, and the fact that as the robot moves, the whole image appears to be in motion given the simple motion test used. The next piece to be added to the system is a heterogeneous blob detector and a region colorer. The blobs are matched to the motion region, and the one with the biggest overlap is used to compute the centroid to be servoed towards. In addition, a binary image consisting of just this blob is latched to replace the motion image for comparison in the next iteration. The result is that now the system uses the motion detector to provide a seed to be tracked over time. To improve the performance of the system, a set of subsumption architecture finite state machines control the hysteresis of switching between blob and motion matching. When the robot is chasing an object, it may lose it for a few frames, in which case motion is re-invoked to try to find a new target to chase; the usual outcome is that the old target is detected again. To an external observer, the robot does not appear to hesitate at all; it is simply pursuing a moving object. A few more subsumption processes take care of special cases, such as accidentally pursuing a wall and running into it.

This vision system worked in a completely unstructured environment. In most images taken by the robot, there was a target object on a dirty floor reflecting highlights form the overhead lights. The system was then augmented further to carry out a quite different task. By taking a homgenous region grower and feeding it into the network at the appropriate point, the robot was made to pursue the floor in front of it, effectively making it wander down corridors.

The conclusion we have reached is that the two architectures can be complementary for navigation. The simple subsumption architecture can underlie simple reflexive behaviors of the insect type. For more sophisticated tasks involving planned visual navigation and recognition, however, the power of a parallel supercomputer is barely sufficient, given the complexity of the tasks.

5. CONCLUSIONS AND FUTURE RESEARCH

Our project, a parallel Vision Machine, has the goal of developing a system for integrating early vision modules and computing a robust description of the discontinuities of the surfaces and of their physical properties that can be used for recognition tasks. During this last year, we have interfaced the output of our integration stage with a parallel model-based recognition algorithm. As we described earlier, the Vision Machine system integrates several vision cues to achieve high performance in unstructured environments, mainly for recognition tasks. It is also a tool for testing our theoretical progress in vision algorithms, their parallel implementation and their integration. The Vision Machine at presents consists of a movable two-camera Eye-Head system - the input device - and a 8K CM2. We are improving the parallel early vision algorithms which compute edge detection, stereo, motion, texture and surface color in close to real-time. The integration stage is based on the technique of coupled Markov Random Field models, and leads to a cartoon-like map of the discontinuities in the scene, with a partial labeling of the brightness edges in terms of their physical origin. In the last year, we have interfaced the output of our integration stage with a parallel model-based recognition algorithm.

The Vision Machine will evolve in several parallel directions:

- Improvement and extensions of its early modules,
- Improvement of the integration and recognition stages (recognition is discussed later),
- Use of the eye-head system in an active mode during recognition task by developing appropriate gaze strategies,
- Use of the results of the integration stage in order to improve the operation of early modules such as stereo and motion by feeding back the preliminary computation of the discontinuities.

Two goals will occupy most of our attention. The first one is the development of the overall organization of the Vision Machine. The system can be seen as an implementation of the *inverse optics* paradigm: it attempts to extract surface properties from the integration of image cues. It must be stressed that we never intended this framework to imply that precise surface properties such as dense, high resolution depth maps, must be delivered by the system. This extreme interpretation of inverse optics seems to be common, but was not the motivation of our project, which originally started with the name *Coarse Vision Machine* to emphasize the importance of computing qualitative, as opposed to very precise, properties of the environment.

Our second main goal in the Vision machine project will be Machine Learning. In particular, we have begun to explore simple learning and estimation techniques for vision tasks. We have succeded in synthetizing a color algorithm from examples (Hurlbert and Poggio, 1988), and in developing a technique to perform unsupervised learning (Sanger, 1988) of other simple vision algorithms such as simple versions of the computation of texture and stereo. In addition, we have used learning techniques to perform integration tasks, such as labeling the type of discontinuities in a scene. We have also begun to explore the connections between recent approaches to learning, such as neural networks, genetic algorithms, and classical methods in approximation theory such as splines, Bayesian techniques and Markov Random Field models. We have identified some common properties of all these approaches and some of the common limitations, such as sample complexity. As a consequence, we now believe that we can leverage our expertise in approximation techniques for the problem of learning in machine vision. Our future theoretical and computational studies will examine available learning techniques, their properties and limitations and develop new ones for the tasks of early vision, for the integration stage and for object recognition. The algorithms will be tested with the Vision Machine system and eventually incorporated into it. We will also pay attention to parallel network implementations of these algorithms: for this subgoal we will be able to leverage the work we are now doing in developing analog VLSI networks for several of the components of the Vision Machine. Towards the goal of achieving much higher flexibility in the Vision Machine we propose to explore (a) the synthesis of vision algorithms from a set of instances and (b) the refinement and tuning of preprogrammed algorithms, such as edge detection, texture discrimination, motion, color and calibration for stereo. We will also develope techniques to estimate parameters of the integration stage. Much of our effort will be focused on the new scheme for visual recognition of 3D objects, whose key component is the automatic learning of a large database of models. We aim to develop a prototype of a flexible vision system that can, in a limited way, learn from experience.

In the following, we outline some of the other directions of future development.

- Labeling the physical origin of edges: computing qualitative surface attributes.
- Saliency, grouping, and segmentation.
- T Junctions: their detection and use in grouping.
- A VLSI Vision Machine.
- Learning and parameter estimation.

6. RELEVANT TECHNICAL REPORTS

Agarwal, A., L. Nekludova, and W. Lim. "A Parallel *o(log n)* Algorithm for Finding Connected Components in Planar Images." *Proc. Intl. Conf. on Parallel Processing*, 783-786, August, 1987.

Bertero, M., T. Poggio, and V. Torre. "Ill-Posed Problems in Early Vision." *Proceedings of the IEEE*, **76**, 869-889, 1988. Also *Artificial Intelligence Memo 909 Center for Biological Information Processing Paper 25*, Massachusetts Institute of Technology, April, 1987.

Blelloch, G. "Scans as Primitive Parallel Operations." *Proc. Intl. Conf. on Parallel Processing*, 355-362, August, 1987.

Blelloch, G., and J. Little "Parallel Solutions to Geometric Problems on the Scan Model of Computation." *Artificial Intelligence Laboratory Memo 952*, Massachusetts Institute of Technology, Cambridge, MA, 1987.

Blelloch, G., and C. Rosenberg. "Network Learning on the Connection Machine." *Proc. Intl. Joint. Conf. on Artificial Intell.*, 323-326, August, 1987.

Brooks, R., and J. Connell. "Navigation Without Representation." *Artificial Intelligence Laboratory Technical Report*, Massachusetts Institute of Technology, Cambridge, MA, in progress.

Brooks, R., and J. Connell. "Asynchronous Distributed Control Systems for a Mobile Robot." *Proc. S.P.I.E.*, 727, October, 1986.

Brooks, R., A. Flynn, and T. Marill. "Self Calibration of Motion and Stereo Vision for Mobile Robot Navigation." *Artificial Intelligence Laboratory Memo 984*, Massachusetts Institute of Technology, Cambridge, MA, August, 1987.

Connell, J. "Task Oriented Spatial Representation for Distributed Systems." *Artificial Intelligence Laboratory Technical Report*, Massachusetts Institute of Technology, Cambridge, MA, September, 1986.

Gamble, E., and T. Poggio. "Visual Integration and Detection of Discontinuities: The Key Role of Intensity Edges." *Artificial Intelligence Laboratory Memo 970/Center for Biological Information Processing Paper 027*, Massachusetts Institute of Technology, September, 1987.

Gillett, W. "Issues in Parallel Stereo Matching." Master's Thesis, Dept. of Brain & Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA, 1988.

Geiger, D., and T. Poggio. "An Optimal Scale for Edge Detection." *Artificial Intelligence Laboratory Memo 1078*, Massachusetts Institute of Technology, Cambridge, MA, September, 1988.

Geiger, D., and T. Poggio. "Level Crossings and the Panum Area." In: *Proceedings IEEE Computer Society Workshop on Computer Vision*. IEEE, Miami, FL, 211-214, December, 1987.

Geiger, D., and T. Poggio. "An Optimal Scale for Edge Detection." In: *Proceedings of the International Joint Conference on Artificial Intelligence*, *Vol. 2*, Milan, Italy, 645-748, 1987.

Grimson, W. "Combinatorics of Object Recognition in Cluttered Environments Using Constrained Search." *Artificial Intelligence*, to appear, 1989. Also *Artificial Intelligence Laboratory Memo 1019*, Massachusetts Institute of Technology, Cambridge, MA, 1988.

Grimson, W. "On the Recognition of Curved Objects in Two Dimensions." *IEEE Pattern Analysis and Machine Intelligence*, to appear, 1988. Also *AIL Memo 983*, Massachusetts Institute of Technology, Cambridge, MA, 1989.

Grimson, W. "On the Recognition of Parameterized 2D Objects." *International Journal of Computer Vision*, accepted for publication, 1988. Also *Artificial Intelligence Laboratory Memo 985*, Massachusetts Institute of Technology, Cambridge, MA, 1988.

Grimson, W. "The Combinatorics of Object Recognition in Cluttered Environments Using Constrained Search." *Proc. Second Intl. Conf. on Computer Vision*, Tarpon Springs, FL, December, 1988.

Grimson, W. "Determining Object Pose for Grasping and Manipulation." In: *Vision and Action: The Control of Grasping*, M. Goodale (ed.), Ablex Publishing Corporation, 1988.

Grimson, W., and D. Huttenlocher. "On the Sensitivity of the Hough Transform for Object Recognition." *Proc. Second Intl. Conf. on Computer Vision*, Tarpon Springs, FL, December, 1988. Also *AIL Memo 1044*, Massachusetts Institute of Technology, Cambridge, MA, May, 1988.

Grimson, W., and T. Lozano-Pérez. "Localizing Overlapping Parts by Searching the Interpretation Tree." *IEEE Pattern Analysis and Machine Intelligence*, **9**, 469-482, 1987. Also *Artificial Intelligence Laboratory Memo 841*, Massachusetts Institute of Technology, Cambridge, MA, 1987.

Heel, J. "Dynamical Systems and Motion Vision." *Artificial Intelligence Laboratory Memo 1037*, Massachusetts Institute of Technology, Cambridge, MA, April, 1988.

Hillis, W.D. **The Connection Machine**, The MIT Press, Cambridge, MA, 1985.

Horswill, I., and R. Brooks. "Situated Vision in a Dynamic World: Chasing Objects." *Proceedings of AAAI*, 796-800, August, 1988.

Hurlbert, A., and T. Poggio. "Synthesizing a Color Algorithm from Examples." *Science*, **27**, 116-120, January, 1988.

Huttenlocher, D., and S. Ullman. "Object Recognition Using Alignment." *Proc. Intl. Conf. on Computer Vision*, 102-111, June, 1987.

Lee, H. "Estimating the Illuminant Color from the Shading of a Smooth Surface." *Artificial Intelligence Laboratory Memo 1068*, Massachusetts Institute of Technology, Cambridge, MA, August, 1988.

Lim, W. "Shape Recognition in the Rocks World." Ph.D. Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, May, 1988.

Lim, W. "Using Occluding Contours for Object Recogition." *Proc. Image Understanding Workshop*, Defense Advanced Research Projects Agency, 909-914, Los Angeles, CA, February, 1987.

Lim, W. "Fast Algorithms for Labelling Connected Components in 2-D Arrays." *TMC Technical Report NA86-1*, Thinking Machines Corp., Cambridge, MA, December, 1986.

Lim, W., A. Agarwal, and L. Nekludova. "A Fast Parallel Algorithm for Labeling Connected Components." *TMC Technical Report NA86-2*, Thinking Machines Corp., Cambridge, MA, December, 1986.

Little, J., G. Blelloch, and T. Cass. "How to Program the Connection Machine for Computer Vision." *Proc. Workshop on Comp. Arch. for Pattern Anal. and Mach. Intell.*, October, 1987.

Little, J., G. Blelloch, and T. Cass. "Parallel Algorithms for Computer Vision on the Connection Machine." *Proc. Intl. Conf. on Computer Vision*, 587-591, June, 1987.

Little, J., G. Blelloch, and T. Cass. "Parallel Algorithms for Computer Vision on the Connection Machine." *Proc. Image Understanding Workshop*, Defense Advanced Research Projects Agency, 628-638, Los Angeles, CA, February, 1987.

Little, J., H. Bülthoff, and T. Poggio. "Parallel Optical Flow Computation." *Proc. Image Understanding Workshop*, Defense Advanced Research Projects Agency, 915-920, Los Angeles, CA, February, 1987.

Little, J., and T. Poggio. "The Vision Machine Project: Integrating Early Vision Modules." In: *Proceedings 1988 Spring Symposium Series - Physical and Biological Approaches to Computational Vision*, AAAI Symposium Series, Stanford, CA, 85-87, March 1988.

Mahoney, J. "Image Chunking: Defining Spatial Building Blocks for Scene Analysis." Master's Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA,

1986. Also *Artificial Intelligence Laboratory Technical Report 980*, Massachusetts Institute of Technology, Cambridge, MA, January, 1987.

Marroquin, S., S. Mitter, and T. Poggio, "Probabilistic Solution of Ill-Posed Problems in Computational Vision." *J. Amer. Stat. Assoc.*, **82**, 76-89, 1987.

Moore, B., and T. Poggio, "Representation of Properties of Multilayer Networks" (abstract). International Neural Network Society Annual Meeting, Boston, September 6-10, 1988.

Poggio, T., and the staff of the Artificial Intelligence Laboratory, "MIT Progress in Understanding Images." In: *Proceedings of the Image Understanding Workshop*, L. Bauman (ed.), Science Applications International Corporation, McLean, VA, 1988.

Poggio, T. and the staff of the A.I. Laboratory, "MIT Progress in Understanding Images." In: *Proceedings of the Image Understanding Workshop*, L. Bauman (ed.), Science Applications International Corporation, McLean, VA, 1988.

Poggio, T. "Learning, Regularization and Splines" (abstract). International Neural Network Society Annual Meeting, Boston, MA, September 6-10, 1988.

Poggio, T. "Computer Vision." In: *Biological and Artificial Intelligence Systems*, E. Clementi and S. Chin, eds., ESCOM Science Publishers, Leiden, The Netherlands, 471-483, 1988.

Poggio, T., E. Gamble, and J. Little, "Parallel Integration of Vision Modules." In: *Proceedings 1988 Spring Symposium Series - Physical and Biological Approaches to Computational Vision*, AAAI Symposium Series, Stanford, CA, 88-95, March 1988.

Poggio, T., E. Gamble, and J. Little, "Parallel Integration of Vision Modules." *Science*, **242**, 436-440 (and cover), October 1988.

Poggio, T., J. Little, E. Gamble, W. Gillett, D. Geiger, D. Weinshall, M. Villalba, N. Larson, T. Cass, H. Bülthoff, M. Drumheller, P. Oppenheimer, W. Yang, and A. Hurlbert, "The Vision Machine." In: *Proceedings of the Image Understanding Workshop*, L. Bauman (ed.), Science Applications International Corporation, McLean, VA, 1988.

Poggio, T., V. Torre, and C. Koch, "Computational Vision and Regularization Theory." In: *Readings in Computer Vision*, M.A. Fischler and O. Firschein (eds.), Morgan Kaufmann Publishers, Los Altos, CA, 1987.

Poggio, T., H. Voorhees, and A. Yuille, "A Regularized Solution to Edge Detection." *Journal of Complexity*, **4**, 106-123, 1988.

Poggio, T., W. Yang, and V. Torre. "Optical Flow: Computational Properties and Networks, Biological and Analog." *The Neuron as a Computational Unit Proceedings*, King's College, Cambridge, UK, June 1988.

Ullman, S., and A. Sha'ashua. "Structural Saliency: The Detection of Globally Salient Structures Using a Locally Connected Network." *Artificial Intelligence Laboratory Memo 1061*, Massachusetts Institute of Technology, Cambridge, MA, July 1988.

Verri, A., and T. Poggio. "Against Quantitative Optical Flow." *Proc. Intl. Conf. on Computer Vision*, 171-180, June, 1987.

Verri, A., and T. Poggio. "Qualitative Information in the Optical Flow." *Proc. Image Understanding Workshop*, Defense Advanced Research Projects Agency, 825-834, Los Angeles, CA, February, 1987.

Voorhees, H. "Finding Texture Boundaries in Images." Master's Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, June, 1987. Also *Artificial Intelligence Laboratory Technical Report 968*, Massachusetts Institute of Technology, Cambridge, MA, June, 1987.

Voorhees, H., and T. Poggio. "Computing Texture Boundaries from Images." *Nature*, **333**, 364-367, 1988.

Voorhees, H., and T. Poggio. "Detecting Textons and Texture Boundaries in Natural Images." *Proc. Intl. Conf. on Computer Vision*, 250-258, Washington, DC, June, 1987.

Voorhees, H., and T. Poggio. "Detecting Blobs as Textons in Natural Images." *Proc. Image Understanding Workshop*, Defense Advanced Research Projects Agency, 892-899, Los Angeles, CA, February, 1987.

Weinshall, D. "Seeing 'Ghost' Solutions in Stereo Vision." *Artificial Intelligence Laboratory Memo 1073; Center for Biological Information ProcessingMemo 33*, Massachusetts Institute of Technology, Cambridge, MA, September, 1988.

Weinshall, D. "Qualitative Depth and Shape from Stereo in Agreement with Psychophysical Evidence." *Artificial Intelligence Laboratory Memo 1007 Center for Biological Information ProcessingMemo 28*, Massachusetts Institute of Technology, Cambridge, MA, December, 1987.

APPENDICES

COVER Camera image overlayed with discontinuities in texture (yellow), motion (orange), and stereo depth (green). The discontinuities are computed from the output of early vision modules coupled with brightness gradient data using Markov random fields. The union of the discontinuities produces a "cartoon" which is used by a parallel recognition algorithm. See page 436. [T. Poggio *et al.*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139]

Parallel Integration of Vision Modules

T. Poggio, E. B. Gamble, J. J. Little

Computer algorithms have been developed for several early vision processes, such as edge detection, stereopsis, motion, texture, and color, that give separate cues to the distance from the viewer of three-dimensional surfaces, their shape, and their material properties. Not surprisingly, biological vision systems still greatly outperform computer vision programs. One of the keys to the reliability, flexibility, and robustness of biological vision systems is their ability to integrate several visual cues. A computational technique for integrating different visual cues has now been developed and implemented with encouraging results on a parallel supercomputer.

Although it is reasonable that combining the evidence provided by multiple visual cues—for example, edge detection, stereo, and color—should provide a more reliable map of the objects in a visual scene than any single cue alone, it is not obvious how to accomplish this integration. One of the most important constraints for recovering surface properties from each of the individual cues is that the physical processes underlying image formation, such as depth, orientation, and reflectance of the surfaces, change slowly in space (adjacent points on a surface are not at random depths, for instance). Standard regularization (*1–3*), on which many examples of the early vision algorithms are based, captures those smoothness properties well. The physical properties of surfaces, however, are smooth almost everywhere, but not at discontinuities. Reliable detection of discontinuities of the physical properties of surfaces is critical for a vision system, since discontinuities are often the most important locations in a scene: depth discontinuities, for example, normally correspond to the boundaries of an object. Thus, the output of each vision module has to be smoothed and interpolated (that is, "filled-in"), since it is noisy and often sparse; at the same time discontinuities must be detected.

Discontinuities can also be used effectively to fuse information between different visual cues (*4–7*) and the image data [see also (*8–10*)]. For instance, a depth discontinuity usually produces a sharp change of brightness in the image (usually called a brightness edge); and a motion boundary often corresponds to a depth discontinuity (and a brightness edge) in the image. The idea is thus to couple different cues—stereo, motion, texture, color, and motion—to the image data (in particular, to the sharp changes of brightness in the image) through the discontinuities in the physical properties of the surfaces (see Fig. 1) [for early work in this direction, see (*11*)]. The final goal of this approach is to use information from several cues simultaneously to refine the initial estimation of surface discontinuities. In this report we will describe a first step in this direction that combines brightness edges with discontinuities in each of the modules separately.

How can this be done? We have chosen to use the machinery of Markov random fields (MRFs), initially suggested for image processing by Geman and Geman (*12*) [for alternative approaches see (*13–16*)]. Consider the prototypical problem of approximating a surface (f) given sparse and noisy data (depth data), on a regular two-dimensional lattice of sites (Fig. 2). We first define the prior probability of the class of surfaces in which we are interested. The probability of a certain depth at any given site in the lattice depends only upon neighboring sites (the Markov property). Because of the Clifford-Hammersley theorem, the prior probability has the Gibbs form:

$$P(f) = \frac{1}{Z}e^{-U(f)/T} \qquad (1)$$

where Z is a normalization constant, T is a

Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.

[redacted] Published in:
Proc. of the Image Understanding
Workshop, L. Bauman (ed.)
SAI Corp., McLean, VA, 1988

THE MIT VISION MACHINE

T. Poggio, J. Little, E. Gamble, W. Gillett, D. Geiger
D. Weinshall, M. Villalba, N. Larson, T. Cass, H. Bülthoff,
M. Drumheller, P. Oppenheimer, W. Yang, and A. Hurlbert

The Artificial Intelligence Laboratory
Massachusetts Institute of Technology

ABSTRACT

We describe the MIT Vision Machine, our goals and achievements to date. The Vision Machine is a computer system that attempts to integrate several vision cues to achieve high performance in unstructured environments for the tasks of recognition and navigation. It is also a test-bed for our theoretical progress in early vision algorithms, their parallel implementation and their integration. The Vision Machine consists of a movable two-camera Eye-Head system - the input device - and a 16K Connection Machine - our main computational engine. We have developed and implemented several parallel early vision algorithms which compute edge detection, stereo, motion, texture and surface color in close to real-time. The integration stage is based on the technique of coupled Markov Random Field models, and leads to a cartoon-like map of the discontinuities in the scene, with a partial labeling of the brightness edges in terms of their physical origin. We will interface the output of our integration stage with available recognition algorithms. We are also beginning to study analog and hybrid VLSI implementations of the Vision Machine main components.

1. Introduction: The Project and Its Goals

Computer vision has developed algorithms for several early vision processes, such as edge detection, stereopsis, motion, texture, and color, which give separate cues as to the distance from the viewer of three dimensional surfaces, their shape, and their material properties. Biological vision systems, however, greatly outperform computer vision programs. It is increasingly clear that one of the keys to the reliability, flexibility and robustness of biological vision systems in unconstrained environments is their ability to integrate many different visual cues. For this reason we are developing a *Vision Machine System* to explore the issue of the integration of early vision modules. The system also serves the purpose of developing parallel vision algorithms since its main computational engine is a parallel supercomputer - the Connection Machine.

The idea behind the Vision Machine is that the main goal of the integration stage is to compute a map of the visible discontinuities in the scene, somewhat similar to a cartoon or a line-drawing. There are several reasons for this. Firstly, experience with existing model-based recognition algorithms suggest that the critical problem in this type of recognition is to obtain a reasonably good map of the scene in terms of features such as edges and corners. The map does not need to be perfect – human recognition works with noisy and occluded line drawings – and of course it cannot be perfect. But it should be significantly cleaner than the typical map provided by an edge detector. Secondly, discontinuities of surface properties are the most important locations in a scene. Thirdly, we have argued [Poggio, 1985] that discontinuities are ideal for integrating information from different visual cues.

It is also clear that there are several different approaches to the problem of how to integrate visual cues. Let us list some of the obvious possibilities:

1) There is no active integration of visual processes. Their individual outputs are "integrated" at the stage at which they are used, for example by a navigation system. This is the approach advocated by Brooks [1987]. While it makes sense for automatic, insect-like, visuo-motor tasks such as tracking a target or avoiding obstacles (e.g., the fly's visuo-motor system [Reichardt and Poggio, 1976]), it seems quite unlikely for visual perception in the wide sense.

2) The visual modules are so tightly coupled that it is impossible to consider visual modules as separate, even in a first order approximation. This view is unattractive on epistemological, engineering and psychophysical grounds.

3) The visual modules are coupled to each other and to the image data in a parallel fashion – each process represented as an array coupled to the arrays associated with the other processes. This point of view is in the tradition of Marr's 2 $\frac{1}{2}$-D sketch, and especially of the "intrinsic images" of Barrow and Tenenbaum [1978]. Our present scheme is of this type, and exploits the machinery of Markov Random Field (MRF) models.

4) Integration of different vision modalities is taking place in a task-dependent way at specific locations – not over the whole image – and when it is needed – therefore not at all times. This approach is suggested by psychophysical data on visual attention and by the idea of visual routines [Ullman, 1984; see also Hurlbert and Poggio, 1986; Mahoney, 1987].

We are presently exploring the third of these approaches. We believe that the last two approaches are compatible with each other. In particular, visual routines may operate on maps of discontinuities such as those delivered by the present Vision Machine, and therefore be located *after* a parallel, automatic integration stage. In real life, of course, it may be more a matter of coexistence. We believe, in fact, that a control structure based on specific knowledge about the properties of the various modules, the specific scene and the specific task will be needed in a later version of the Vision Machine to overview and control the MRF integration stage itself and its parameters. It is possible that the integration stage should be much more goal-directed that what our present methods (MRF based) allow. The main goal of our work is to find out whether this is true.

The Vision Machine project has a number of other goals. It provides a focus for developing parallel vision algorithms and for studying how to organize a real-time vision system on a massively parallel supercomputer. It attempts to change the usual paradigm of computer vision research over the past years: choose a specific problem, for example stereo, find an algorithm, and test it in isolation. The Vision Machine allows us to develop and test an algorithm in the context of the other modules and the requirements of the overall visual task – above all visual recognition. For this reason, the project is more than an experiment in integration and parallel processing: it is a laboratory for our theories and algorithms.

Finally, the goal of the Vision Machine project is no less than the ultimate goal of vision research: to build a vision system that achieves human-level performance.

2. The Vision Machine System

The overall organization of the system is shown in Figure 1. The image(s) are processed through independent algorithms or modules corresponding to different visual cues, in parallel. Edges are extracted using Canny's edge detector. Stereo computes disparity from the left and right images. The motion module estimates an approximation to the optical flow from pairs of images in a time sequence. The texture module computes texture attributes (such as density and orientation of textons [see Voorhees, 1987]). The color algorithm provides an estimate of the spectral albedo of the surfaces, independently of the *effective illumination*, that is, illumination gradients and shading effects, as suggested by Hurlbert and Poggio [see Poggio, 1985].

The measurements provided by the early vision modules are typically noisy and possibly sparse (for stereo and motion). They are smoothed and made dense by exploiting known constraints within each process (for instance, that disparity is smooth). This is a stage of *approximation* and *restoration* of data, performed by using a Markov Random Field model. Simultaneously, discontinuities are found in each cue. Prior knowledge of the behavior of discontinuities is exploited, for instance, the fact that they are continuous lines, not isolated points. Detection of discontinuities is aided by the information provided by brightness edges. Thus each cue – disparity, optical flow, texture, and color – is coupled to the edges in brightness.

The full scheme involves finding the various types of physical discontinuities in the surfaces – *depth discontinuities (extremal edges and blades), orientation discontinuities, specular edges, albedo edges (or marks), shadow edges* – and coupling them with each other and back to the discontinuities in the visual cues, as illustrated in Figure 1. So far we have implemented only the coupling of brightness edges to each of the cues provided by the early algorithm. As we will discuss later, the technique we used to approximate, to simultaneously detect discontinuities, and to couple the different processes, is based on MRF models. The output of the system is a set of labeled discontinuities of the surfaces around the viewer. In our implemented version of the system we find discontinuities in disparity, motion, texture, and color. These discontinuities, taken together, represent a "cartoon" of the original scene which can be used for recognition and navigation (along with, if

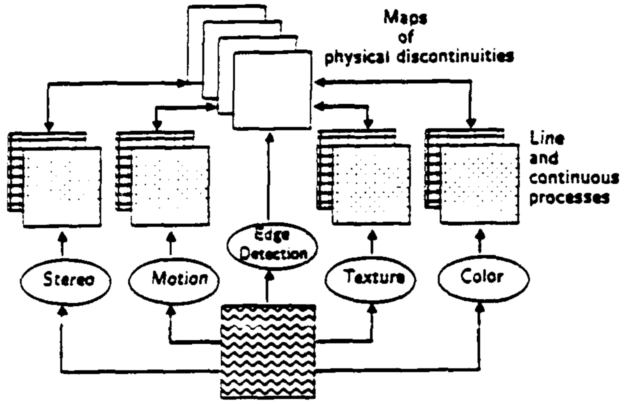


Figure 1: Block Diagram of the Vision Machine

needed, interpolated depth, motion, texture and color fields).

The plan of the paper is as follows. We will first review the present hardware of the Vision Machine: the Eye-Head system and the Connection Machine. We will then describe in some detail each of the early vision algorithms that are presently running and are part of the system. After this, the integration stage will be discussed. We will analyze some results and illustrate the merits and the pitfalls of our present system. The last chapter will discuss a real-time visual system and some ideas on how to put the system into VLSI circuits of analog and digital type.

3. Hardware

3.1. The Eye-Head System

Because of the variety of visual information processed by the Vision Machine, a general purpose image input device is required. Such a device is the Eye-Head system. Here we discuss its current and future configurations.

3.1.1. The Present

The Eye-Head system (Figure 2a) consists of two CCD cameras ("eyes") mounted on a variable-attitude platform ("head"). The apparatus allows the cameras to be moved as a unit, analogous to head movement. It also allows the lines of sight of the cameras to be pointed independently, analogous to eye movement. Each camera is equipped with a motorized zoom lens ($F1.4$, focal length from 12.5 to 75mm), allowing control of the iris, focus, and focal length by the host computer (currently a Symbolics 3600 Lisp Machine). Other hardware allows for repeatable calibration of the entire apparatus.

Because of the size and weight of the motorized lenses, it would be impractical to achieve eye movement by pointing the camera/lens assemblies directly. Instead, each assembly is mounted rigidly on the head, with eye movement achieved indirectly. In front of each camera lens is a pair of front surface mirrors (Figure 2b), each of which can be pivoted by a galvanometer also mounted rigidly on the head. The mirrors are positioned to provide two degrees of freedom in aiming the cameras. At the expense of a more complicated imaging geometry, this allows for a simpler and faster control system for the eyes.

The head is attached to its mount via a spherical joint, allowing head rotation about two orthogonal axes (pan and tilt). Each axis is driven by a stepper motor coupled to the drive shaft through a harmonic drive. The latter provides a large gear ratio in conjunction with very little mechanical backlash. Under control of the stepper motors, the head can be panned 180 degrees from left to right, and tilted 90 degrees (from vertical-down to horizontal). Each of the stepper motors is provided with an optical shaft encoder for shaft position feedback (a closed-loop control scheme is employed for the stepper motors). The shaft encoders also provide an index pulse (one per revolution) which is used for joint calibration in conjunction with mechanical limit switches. The latter also protect the head from damage due to excessive travel.

The overall control system for the Eye-Head system is distributed over a micro-processor network (UNET) developed at the MIT AI Lab for the control of vision/robotics hardware. The UNET is a "multi-drop" network supporting up to 32 micros, under the control of a single host. The micros normally function as network slaves, with the host acting as the master. In this mode the micros only "speak when spoken to", responding to various network operations either by receiving information (command or otherwise) or by transmitting information (such as status or results). Associated with each micro on the UNET is a local 16-bit bus (UBUS), which is totally under the control of the micro. Peripheral devices such as motor drivers, galvanometer drivers, and pulse width modulators (PWMs), to name a few, can be interfaced at this level.

At present two micro-processors are installed on the Eye-Head UNET: one for the galvanometer and one for both the motorized lenses and stepper motors. The processors currently employed are based on the Intel 8051. Each of these micros has an assortment of UBUS

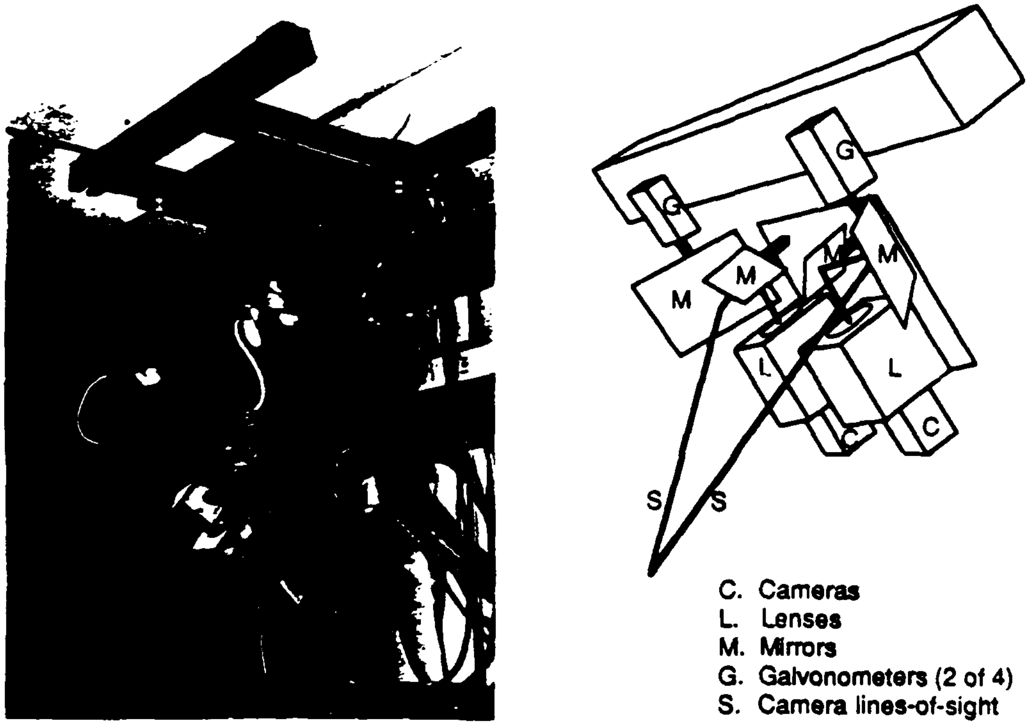


Figure 2: The Eye-Head System

peripherals under its control. By making these peripherals sufficiently powerful, each micro's control task can remain simple and manageable. Code for the micros, written in both assembly language and C, is facilitated by a Lisp-based debugging environment.

3.1.2. The Future

A single enhancement remains for the Eye-Head system. Currently, a Symbolics Lisp Machine acts as the host processor for the UNET. Soon an intermediate real-time processor will be placed between the Lisp Machine and the UNET, acting as master of the latter. The real-time processor (referred to as the DSP, being based on a Digital Signal Processor chip) will relieve the Lisp Machine of all the UNET protocol tasks, as well as various low-level, real-time control tasks for which the Lisp Machine is ill-suited. Among the tasks envisioned for the DSP is optimal position estimation of moving targets from motion data.

3.2. Our Computational Engine: The Connection Machine

The Connection Machine is a powerful fine-grained parallel machine which has proven useful for implementation of vision algorithms. In implementing these algorithms, several different models of using the Connection Machine have emerged, since the machine provides several different communication modes. The Connection Machine implementation of algorithms can take advantage of the underlying architecture of the machine in novel ways. We describe here several common, elementary operations which recur throughout the following discussion of parallel algorithms.

3.2.1. The Connection Machine

The CM-1 version of the Connection Machine [Hillis, 1985] is a parallel computing machine with between 16K and 64K processors, operating under a single instruction stream broadcast to all processors. It is a Single Instruction Multiple Data (SIMD) machine; all processors execute the same control stream. Each processor is a simple 1-bit processor, currently with 4K bits of memory. There are two modes of communication among the processors: first, they are connected by a mesh into a 128 × 512 grid network (the NEWS network, so-called because the connections are in the four cardinal directions), allowing rapid direct communication between neighboring processors, and second, the *router*, which allows messages to be sent from any processor to any other processor in the machine. The processors in the Connection Machine can be envisioned as being the vertices of a 16-dimensional hypercube (in fact, it is a 12-dimensional hypercube; at each vertex of the hypercube resides a chip containing 16 processors). Each processor in the Connection Machine is identified by its hypercube address in the range 0 . . . 65535, imposing a linear order on the processors. This address denotes the destination of messages handled by the router. Messages pass along the edges of the hypercube from source processors to destination processors. The Connection Machine also has facilities for returning to the host machine the result of various operations on a field in all processors; it can return the global maximum, minimum, sum, logical AND, and logical OR of the field.

To allow the machine to manipulate data structures with more than 64K elements, the Connection Machine supports *virtual processors.* A single physical processor can operate as a set of multiple virtual processors by serializing operations in time, and partitioning the memory of each processor. This is otherwise invisible to the user. Connection Machine programs utilize Common Lisp syntax, in a language called *Lisp, and are manipulated in the same fashion as Lisp programs.

3.2.2. Powerful Primitive Operations

Many vision problems must be solved by a combination of communication modes on the Connection Machine. The design of these algorithms takes advantage of the underlying architecture of the machine in novel ways. There are several common, elementary operations used in this discussion of parallel algorithms: routing operations, scanning and distance doubling.

Routing

Memory in the Connection Machine is associated with processors. Local memory can be accessed rapidly. Memory of processors nearby in the NEWS network can be accessed by passing it through the processors on the path between the source and the destination. At present, NEWS accesses in the machine are made in the same direction for all processors. The *router* on the Connection Machine provides parallel reads and writes among processor memory at arbitrary distances and with arbitrary patterns. It uses a packet-switched message routing scheme to direct messages along the hypercube connections to their destinations. This powerful communication mode can be used to reconfigure completely, in one parallel write operation taking one router cycle, a field of information in the machine. The Connection Machine supplies instructions so that many processors can read from the same location or write to the same location, but since these memory references can cause significant delay, we will usually only consider exclusive read, exclusive write instructions. We will usually not allow more than one processor to access the memory of another processor at one time. The Connection Machine can combine messages at a destination by various operations, such as logical AND, inclusive OR, summation, and maximum or minimum.

```
processor-number = [0  1  2  3  4  5  6  7]
A                = [5  1  3  4  3  9  2  6]
Plus-Scan(A)     = [5  6  9 13 16 25 27 33]
Max-Scan(A)      = [5  5  5  5  5  9  9  9]
```

Figure 3: Examples of *Plus-Scan* and *Max-Scan*.

Scanning

The *scan* operations [Blelloch, 1987] can be used to simplify and speed up many algorithms. They directly take advantage of the hypercube connections underlying the router, and can be used to distribute values among the processors and to aggregate values using associative operators. Formally, the *scan* operation takes a binary associative operator $\oplus$, with identity 0, and an ordered set $[a_0, a_1, \ldots, a_{n-1}]$, and returns the set $[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})]$. This operation is sometimes referred to as the *data independent prefix operation* [Kruskal et.al., 1985]. Binary associative operators include *minimum, maximum, and plus.* Figure 3 shows scans using *maximum* and *plus.*

The four scan operations *plus-scan*, *max-scan*, *min-scan* and *copy-scan* are implemented in microcode and take about the same amount of time as a routing cycle. The *copy-scan* operation takes a value at the first processor and distributes it to the other processors. These scans operations can take *segment bits* that divide the processor ordering into segments. The beginning of each segment is marked by a processor whose segment bit is set, and the scan operations start over again at the beginning of each segment (see Figure 4).

The *scan* operations also work using the NEWS addressing scheme, termed *grid-scans.* These compute the sum, and find the maximum, copy, or number values along rows or columns of the NEWS grid quickly.

For example, *grid-scans* can be used to find for each pixel the sum of a square region with width $2m+1$ centered at the pixel. This sum is computed using the following steps. First, a *plus-scan* accumulates partial sums for all pixels along the rows. Each pixel then gets the result of the scan from the processor m in front of it and m behind it; the difference of these two values represents the sum, for each pixel, of its neighborhood

```
processor-number    = [0  1  2  3  4  5  6  7]
A                   = [5  1  3  4  3  9  2  6]
SB (segment bit)    = [1  0  1  0  0  0  1  0]
Max-Scan(A, SB)     = [5  5  3  4  4  9  2  6]
Copy-Scan(A, SB)    = [5  5  3  3  3  3  2  2]
Plus-Scan(A, SB)    = [0  5  6  3  7 10 19  2]
Min-Scan(A, SB)     = [MX 5  1  3  3  3  3  2]
```

Figure 4: Examples of Segmented *Scan* Operations.

along the row. We now execute the same calculation on the columns, resulting in the sum, for each pixel, of the elements in its square. The whole process only requires a few *scans* and routing operations, and runs in time independent of the size of m. The summation operations are generally useful to accumulate local support in many of our algorithms, such as stereo and motion.

Distance Doubling

Another important primitive operation is *distance doubling* [Wyllie, 1979; Lim, 1986], which can be used to compute the effect of any binary, associative operation, as in *scan*, on processors linked in a list or a ring. For example, using *max*, *doubling* can find the extremum of a field contained in the processors. Using message-passing on the router, *doubling* can propagate the extreme value to all processors in the ring of N processors in $O(\log N)$ steps. Each step involves two *send* operations. Typically, the value to be maximized is chosen to be the hypercube-address. At termination, each processor in the ring knows the label of the maximum processor in the ring, hereafter termed the *principal processor*. This labels all connected processors uniquely and nominates a processor as the representative for the entire set of connected processcrs. At the same time, the distance from the *principal* can be computed in each processor. Figure 4 shows the propagation of values in a ring of eight processors. Each processor initially, at step 0, has the address of the next processor in the ring, and a value which is to be maximized. At the termination of the i^{th} step, a processor knows the addresses of processors $2^i + 1$ away and the maximum of all values within 2^{i-1} processors away. In the example, the maximum value has been propagated to all 8 processors in $\log 8 = 3$ steps.

4. Early Vision Algorithms and their Parallel Implementation

4.1. Edge Detection

Edge detection is a key first step in correctly identifying physical changes. The apparently simple problem of measuring sharp brightness changes in the image has proven to be difficult. It is now clear that edge detection should be intended not simply as finding "edges" in the images, an ill-defined concept in general, but as measuring appropriate derivatives of the brightness data. This involves the task-dependent use of different two-dimensional derivatives. In many cases, it is appropriate to mark locations corresponding to appropriate critical points of the derivative such as maxima or zeroes. In some cases, later algorithms based on these binary features – presence or absence of edges – may be equivalent, or very similar, to algorithms that directly use the continuous value of the derivatives. A case in point is provided by our stereo and motion algorithms, to be described later. As a consequence, one should not always make a sharp distinction between edge-based and intensity based algorithms: the distinction is more blurred and in some cases, it is almost a matter of implementation.

In our current implementation of the Vision Machine, we are using two different kinds of edges. The first consists of zero-crossings in the Laplacian of the image filtered through an appropriate Gaussian. The second consists of the edges found by Canny's edge detector. Zero-crossings can be used by our stereo and motion algorithms (though we have mainly used Canny's edges at fine resolution). Canny's edges (at a coarser resolution) are input to the MRF integration scheme.

Zero-Crossings

Because the derivative operation is ill-posed, we need to filter the resultant data through an appropriate low-pass filter [Torre and Poggio, 1985]. The filter of choice (but not the only possibility!) is a Gaussian at a suitable spatial scale. An interesting, simple implementation of Gaussian convolution relies on the binomial approximation to the Gaussian distribution. This algorithm requires only integer addition, shifting, and local communication on the 2-D mesh, so it can be implemented on a simple 2-D mesh architecture (such as the NEWS network on the Connection Machine).

The Laplacian of a Gaussian is often approximated by the difference of Gaussians. The Laplacian of a Gaussian can also be computed by convolution with a Gaussian followed by convolution with a discrete Laplacian; we have implemented both on the Connection Machine. To detect zero-crossings, the computation at each pixel need only examine the sign bits of neighboring pixels.

Canny Edge Detection

The Canny edge detector is often used in image understanding. It is based on directional derivatives, so it has improved localization. The Canny edge detector on the Connection Machine consists of the following steps:

- Gaussian smoothing
- Directional derivative
- Non-maximum suppression
- Thresholding with hysteresis.

Gaussian filtering, as described above, is a local operation. Computing directional derivatives is also local, using a finite difference approximation referencing only local neighbors in the image grid.

Non-maximum Suppression

Non-maximum suppression selects as edge candidates those pixels for which the gradient magnitude is maximal in the direction of the gradient. This involves interpolating the gradient magnitude between each of two pairs of adjacent pixels among the eight neighbors of a pixel, one forward in the gradient direction, one backward. However, it may not be critical to use interpolation, in which case magnitudes of neighboring values can be directly compared.

Thresholding with Hysteresis

Thresholding with hysteresis eliminates weak edges due to noise, using the threshold, while connecting extended curves over small gaps using hysteresis. Two thresholds are computed, *low* and *high*, based on an estimate of the noise in the image brightness. The non-maximum suppression step selects those pixels where the gradient magnitude is maximal in the direction of the gradient. In the thresholding step, all selected pixels with gradient magnitude below *low* are eliminated. All pixels with values above *high* are considered as edges. All pixels with values between *low* and *high* are edges if they can be connected to a pixel above *high* through a chain of pixels above *low*. All others are eliminated.

This is a spreading activation operation; it propagates information along a set of connected edge pixels. The algorithm iterates, in each step marking as *edge* pixels any *low* pixels adjacent to *edge* pixels. When no pixels change state, the iteration terminates, taking $O(m)$ steps, a number proportional to the length m of the longest chain of *low* pixels which eventually become *edge* pixels. The running time of this operation can be reduced to $O(\log m)$, using *distance doubling*.

Noise Estimation

Estimating noise in the image can be performed by analyzing a histogram of the gradient magnitudes. Most computational implementations of this step perform a global analysis of the gradient magnitude distribution, which is essentially non-local; we have had success with a Connection Machine implementation using local histograms. The thresholds used in Canny edge detection depend on the final task for which the edges are used. A conservative strategy can use an arbitrary low threshold to eliminate the need for the costly processing required to accumulate a histogram. Where a more precise estimate of noise is needed, it may be possible to find a scheme that use a coarse estimate of the gradient magnitude distribution, with minimal global communication.

4.2. Stereo

The Drumheller-Poggio parallel stereo algorithm [Drumheller and Poggio, 1986] runs as part of the Vision Machine. Disparity data produced by the algorithm comprise one of the inputs to the MRF-based integration stage of the Vision Machine. We are exploring various extensions of the algorithm, as well as the possible use of feedback from the integration stage. In this section, we will review the algorithm briefly, then proceed to a discussion of current research.

The stereo algorithm runs on the Connection Machine system with good results on natural scenes in times that are typically on the order of one second. The stereo algorithm is presently being extended in the context of the Vision Machine project.

4.2.1. Drumheller-Poggio Stereo Algorithm

Stereo matching is an ill-posed problem [see Bertero et.al., 1987] that cannot be solved without taking advantage of natural constraints. The *continuity constraint* [see, for instance, Marr and Poggio, 1976] asserts that the world consists primarily of piecewise smooth surfaces. If the scene contains no transparent objects, then the *uniqueness constraint* applies: there can be only one match along the left or right lines of sight. If there are no narrow occluding objects, the *ordering constraint* [Poggio and Yuille, 1984] holds: any two points must be imaged in the same relative order in the left and right eyes.

The specific *a priori* assumption on which the algorithm is based is that the disparity – that is, the depth of the surface – is locally constant in a small region surrounding a pixel. It is a restrictive assumption which, however, may be a satisfactory *local* approximation in many cases (it can be extended to more general surface assumptions in a straightforward way but at high computational cost). Let $E_L(x,y)$ and $E_R(x,y)$ represent the left and the right image of a stereo pair or some transformation of it, such as filtered images or a map of the zero-crossings in the two images (more generally, they can be maps containing a feature vector at each location (x, y) in the image).

We look for a discrete disparity $d(x, y)$ at each location x, y in the image that minimizes

$$\|E_L(x, y) - E_R(x + d(x, y), y)\|_{\mathbf{patch}_i} \qquad (1)$$

where the norm is a summation over a local neighborhood centered at each location (x, y); $d(x)$ is assumed constant in the neighborhood. Equation (1) implies that we should look at each (x, y) for $d(x, y)$ such that

$$\int_{\mathbf{patch}_i} (E_L(x, y) E_R(x + d(x, y), y))^2 dx dy \qquad (2)$$

is maximized.

The algorithm that we have implemented on the Connection Machine is actually somewhat more complicated, since it involves geometric constraints that affect the way the maximum operation is performed (see Drumheller and Poggio, 1986). The implementation currently used in the Vision Machine at the AI Lab uses the maps of Canny's edges obtained from each image for E_L and E_R.

In more detail, the algorithm is composed of the following steps:

1) Compute features for matching.

2) Compute potential matches between features.

3) Determine the degree of continuity around each potential match.

4) Choose correct matches based on the constraints of continuity, uniqueness, and ordering.

Potential matches between features are computed in the following way. Assuming that the images are registered so that the epipolar lines are horizontal, the stereo matching problem becomes one-dimensional: an edge in the left image can match any of the edges in the corresponding horizontal scan line in the right image. Sliding the right image over the left image horizontally, we compute a set of *potential match planes*, one for each horizontal disparity. Let $p(x, y, d)$ denote the value of the (x, y) entry of the potential match plane at disparity d. We set $p(x, y, d) = 1$ if there is an edge at location (x, y) in the left image and a compatible edge at location $(x - d, y)$ in the right image; otherwise, set $p(x, y, d) = 0$. In the case of the DOG edge detector, two edges are compatible if the sign of the convolution for each edge is the same.

To determine the degree of continuity around each potential match (x, y, d), we compute a *local support score* $s(x, y, d) =_{Patch} p(x, y, d)$, where *patch* is a small neighborhood of (x, y, d) within the *dth* potential match plane. In effect, nearby points in *patch* can "vote" for the disparity d. The score $s(x, y, d)$ will be high if the continuity constraint is satisfied near (x, y, d), i.e., if *patch* contains many votes. This step corresponds to the integral over the patch in Equation (2).

Finally, we attempt to select the correct matches by applying the uniqueness and ordering constraints (see above). To apply the uniqueness constraint, each match suppresses all other matches along the left and right lines of sight with weaker scores. To enforce the ordering constraint, if two matches are not imaged in the same relative order in left and right views, we discard the match with the smaller support score. In effect, each match suppresses matches with lower scores in its forbidden zone [Poggio and Yuille, 1984]. This step corresponds to choosing the disparity value that maximizes the integral of Equation (2).

4.2.2. Improvements

Using this algorithm as a base, we are exploring the following topics:

Detection of Depth Discontinuities

The Marr-Poggio continuity constraint is both a strength and a weakness of the stereo algorithm. Favoring continuous disparity surfaces reduces the solution space tremendously, but also tends to smooth over depth discontinuities present in the scene. Consider what happens near a linear depth discontinuity, say a point near the edge of a table viewed from above. The square local support neighborhood for the point will be divided between points on the table and points on the floor; thus, almost half of the votes will be for the wrong disparity.

One solution to this problem is feedback from the MRF integration stage. We can take the depth discontinuities located by the integration stage (using the results from a first pass of the stereo algorithm, among other inputs) and use them to restrict the local support neighborhoods so that they do not span discontinuities. In the example mentioned above, the support neighborhood would be trimmed to avoid crossing the discontinuity between the table and the floor, and thus would not pick up spurious votes from the floor.

We can also try to locate discontinuities by examining intermediate results of the stereo algorithm.

Consider a histogram of votes vs. disparity for the table/floor example. For a support region centered near the edge of the table, we expect to see two strong peaks: one at the disparity of the floor, and the other at the disparity of the table. Therefore a bimodal histogram is strong evidence for the presence of a discontinuity.

These two ideas can be used in conjunction. Discontinuity detection within stereo can take advantage of the extra information provided by the vote histograms. By passing better depth data (and perhaps candidate discontinuity locations) to the integration stage, we improve the detection of discontinuities at the higher level.

Improving the Stereo Matcher

The original Drumheller-Poggio algorithm matched DOG zero-crossings, where the local support score counted the number of zero-crossings in the left image patch matching edges in the right image patch, at a given disparity. We have modified the matcher in a variety of ways.

1) Canny edges. The matcher now uses edges derived by a parallel implementation of the Canny edge detector [Canny, 1983; Little et al., 1987] rather than DOG zero-crossings, for better localization.

2) Gradient direction constraint. We allow two Canny edges to match only if the associated brightness gradient directions are aligned within a parameterized tolerance. This is analogous to the restriction in the Marr-Poggio-Grimson stereo algorithm [Grimson,1981], where two zero-crossings can match only if the directions of the DOG gradients are approximately equal. Matching gradient orientations is a tighter constraint than matching the sign of the DOG convolution. Furthermore, the DOG sign is numerically unstable for horizontally oriented edges.

3) The scores are now normalized to take into account the number of edges in the left and right image patches eligible to match, so that patches with high edge densities do not generate artificially high scores. We plan to change the matcher so that edges that fail to match would count as negative evidence by reducing the support score, but this has not yet been implemented.

In the near future, we will explore matching brightness values as well as edges, using a cross-correlation approach similar to that of Little, Bülthoff and Poggio [1987] for motion estimation [Gillett, in preparation].

Identifying Areas that are Outside of the Matcher's Disparity Range

The stereo algorithm searches a limited disparity range, selected manually. Every potential match in the scene (an edge with a matching edge at some disparity) is assigned the in-range disparity with the highest score, even though the correct disparity may be out of range. How can we tell when an area of the scene is out of range?

The most effective approach that we have attempted to date is to look for regions with low matching scores. Two patches that are incorrectly matched will, in general, produce a low matching score.

4.2.3. Memory-Based Registration and Calibration

Registration of the image pair for the stereo algorithm is done by presenting to the system a pattern of dots, roughly on a sparse grid, at the distance around which stereo has to operate. The registration is accomplished using a warping computed by matching the dots from the left and right images. The dots are sparse enough that matching is unambiguous. The matching defines a warping vector for each dot; at other points the warping is computed by bilinear interpolation of the two components of warping vectors. The warping necessary for mapping the right image onto the left image is then stored. Prior to stereo-matching, the right image is warped according to the pre-stored addresses by sending each pixel in the right image to the processor specified in the table.

The warping table corrects for deformations, including those due to vertical disparities and rotations, those due to the image geometry (errors in the alignment of the cameras, perspective projection, errors introduced by the optics, etc.) We plan to store several warping tables for each of a few convergence angles of the two cameras (assuming symmetric convergence). We conjecture that simple interpolation can yield sufficiently accurate warping tables for fixation angles intermediate to the ones stored. Notice that these tables are independent of the position of the head. Absolute depth is not the concern here (we are not using it in our present Vision Machine), but it could easily be recovered from knowledge of the convergence angle. Notice also that the whole registration scheme has the flavor of a learning process. Convergence angles are inputs and warping tables are the outputs of the modules; the set of angles, together with the associated warping tables, represent the set of input-output examples. The system can "generalize" by interpolating between warping tables and providing the warping corresponding to a vergence angle that does not appear in the set of "examples". Calibration of disparity to depth could be done in a similar way.

4.3 Motion

The motion algorithm computes the optical flow field, a vector field which approximates the projected motion field. The procedure produces sparse or dense output, depending on whether it uses edge features or intensities. The algorithm assumes that image displacements are small, within a range $(\pm\delta, \pm\delta)$. It is also assumed that the optical flow is locally constant in a small region surrounding a point. This assumption is strictly only true for translational motion of 3–D planar surface patches parallel to the image plane. It is a restrictive assumption which, however, may be a satisfactory *local* approximation in many cases. Let $E_t(x,y)$ and $E_{t+\Delta t}(x,y)$ represent transformations of two discrete images separated by time interval Δt, such as filtered images or a map of the brightness changes in the two images (more generally, they can be maps containing a feature vector at each location (x,y) in the image) [Kass, 1986; Nishihara, 1984].

We look for a discrete motion displacement $\underline{v} = (v_x, v_y)$ at each location x, y in the image that minimizes

$$||E_t(x,y) - E_{t+\Delta t}(x + v_x\Delta t, y + v_y\Delta t)||_{\text{patch}_i} = \min \quad (3)$$

where the norm is a summation over a local neighborhood centered at each location (x,y); $\underline{v}(x,y)$ is assumed constant in the neighborhood. Equation (3) implies that we should look at each (x,y) for $\underline{v} = (v_x, v_y)$ such that

$$\int_{\text{patch}_i} (E_t(x,y) - E_{t+\Delta t}(x + v_x\Delta t, y + v_y\Delta t))^2 dx dy \quad (4)$$

is minimized. Alternatively, one can maximize the negative of the integrated result. Equation (4) represents the sum of the pointwise squared differences between a patch in the first image centered around the location (x,y) and a patch in the second image centered around the location $(x + v_x\Delta t, y + v_y\Delta t)$.

This algorithm can be translated easily into the following description. Consider a network of processors representing the result of the integrand in Equation (4). Assume for simplicity that this result is either 0 or 1 (this is the case if E_t and $E_{t+\Delta t}$ are binary feature maps). The processors hold the result of differencing (taking the logical "exclusive or") the right and left image map for different values of (x,y) and v_x, v_y. The next stage, corresponding exactly to the integral operation over the patch, is for each processor to summate the total (2) in an (x,y) neighborhood at the same disparity. Note that this summation operation is efficiently implemented on the Connection Machine using *scan* computations. Each processor thus collects a vote indicating support that a patch of surface exists at that displacement. The algorithm iterates over all displacements in the range $(\pm\delta, \pm\delta)$, recording the values of the integral (2) for each displacement. The last stage is to choose $\underline{v}(x,y)$ among the displacements in the allowed range that maximizes the integral. This is done by an operation of "non–maximum suppression" across velocities out of the finite allowed set: at the given (x,y), the processor is found that has the maximum vote. The corresponding $\underline{v}(x,y)$ is the velocity of the surface patch found by the algorithm. The actual implementation of this scheme can be simplified so that the "non–maximum suppression" occurs during iteration over displacements, so that no actual table of summed differences over displacements need be constructed. In practice, the algorithm has been shown to be effective both for synthetic and natural images using different types of features or measurements on the brightness data, including edges (both zero-crossings of the Laplacian of Gaussian and Canny's method), which generate sparse results along brightness edges, or brightness data directly, or the Laplacian of Gaussian or its sign, which generate dense results. Because the optical flow is computed from quantities integrated over the individual patches, the results are robust against the effects of uncorrelated noise.

The comparison stage employs patchwise cross-correlation, which exploits local constancy of the optical flow (the velocity field is guaranteed to be constant for translations parallel to the image plane of a planar surface patch; it is a cubic polynomial for arbitrary motion of a planar surface [see Waxman, 1986; Little et.al., 1987]. Experimentally, we have used zero-crossings, the Laplacian of Gaussian filtered image, its sign, and the smoothed brightness values, with similar results. It is interesting that methods *superficially* so different (edge-based and intensity-based) give such similar results. As we mentioned earlier, this is not surprising. There are theoretical arguments that support, for instance, the equivalence of cross-correlating the sign bit of the Laplacian filtered image and the Laplacian filtered image itself. The argument is based on the following theorem [see Little, Bülthoff and Poggio, in preparation], which is a slight reformulation of a well-known theorem.

Theorem

If $f(x,y)$ and $g(x,y)$ are zero mean jointly normal processes, their cross-correlation is determined fully by the correlation of the sign of f and of the sign of g (and determines it). In particular

$$R_{\tilde{f},\tilde{g}} = \frac{2}{\pi} arcsin(R_{f,g})$$

where $\tilde{f} = sign\ f$ and $\tilde{g} = sign\ g$

Thus, cross-correlation of the sign bit is exactly equivalent to cross-correlation of the signal itself (for Gaussian processes). Notice that from the point of view of information, the sign bit of the signal is completely equivalent to the zero-crossing of the signal. Nishihara first used patchwise cross-correlation of the sign bit of DOG filtered images [Nishihara, 1984], and has implemented it more recently on real-time hardware [Nishihara et.al., 1988].

The existence of discontinuities can be detected in optical flow, as in stereo, both during computation and by processing the resulting flow field. The latter field is input to the MRF integration stage. During computation, discontinuities in optical flow arising from occlusions are indicated by low normalized scores for the chosen displacement.

4.4. Color

The color algorithm that we have implemented is a very preliminary version of a module that should find the boundaries in the surface spectral reflectance function, that is, discontinuities in the surface color. The algorithm relies on the idea of ***effective illumination*** and on the ***single source*** assumption, both introduced by Hurlbert and Poggio [see Poggio et.al., 1985].

The single source assumption states that the illumination may be separated into two components, one dependent only on wavelength and one dependent only on spatial coordinates, and generally holds for illumination from a single light source. It allows us to write the image irradiance equation for a Lambertian world as

$$I^{\nu} = k^{\nu} E(x,y) \rho^{\nu}(x,y) \qquad (5)$$

where I^{ν} is the image irradiance in the νth spectral channel ($\nu = red, green, blue$), $\rho^{\nu}(x,y)$ is the surface spectral reflectance (or albedo) and the effective illumination $E(x,y)$ absorbs the spatial variations of the illumination and the shading due to the 3D shape of surfaces (k^{ν} is a constant for each channel and depends only on the luminant). A simple segmentation algorithm is then obtained by considering the equation

$$H(x,y) = \frac{I^r}{I^r + I^g} = \frac{k^r \rho^r}{k^r \rho^r + k^g \rho^g} \qquad (6)$$

which changes only when ρ^r or ρ^g or both change. Thus H, which is piecewise constant, has discontinuities that mark changes in the surface albedo, independently of changes in the effective illumination.

The quantity $H(x,y)$ is defined almost everywhere, but is typically noisy. To counter the effect of noise, we exploit the prior information that H should be piecewise constant with discontinuities that are themselves continuous, non-intersecting lines. As we will discuss later, this restoration step is achieved by using a MRF model. This algorithm works only under the restrictive assumption that specular reflections can be neglected. Hurlbert [1988] discusses in more detail the scheme outlined here and how it can be extended to more general conditions.

4.5. Texture

The texture algorithm is a greatly simplified parallel version of the texture algorithm developed by Voorhees and Poggio [1987]. Texture is a scalar measure computed by summation of texton densities over small regions surrounding every point. Discontinuities in this mesure can correspond to occlusion boundaries, or orientation discontinuities, which cause foreshortening. Textons are computed in the image by simple approximation to the methods presented in Voorhees and Poggio [1987]. For this example, the textons are restricted to blob-like regions, without regard to orientation selection.

To compute textons, the image is first filtered by a Laplacian of Gaussian filter at several different scales. The smallest scale selects the textural elements. The Laplacian of Gaussian image is then thresholded at a non-zero value to find the regions which comprise the blobs identified by the textons. The result is a binary image with non-zero values only in the areas of the blobs. A simple summation counts the density of blobs, the portion of the summation region covered by blobs, in a small area surrounding each point. This operation effectively measures the density of blobs at the small scale, while also counting the presence of blobs caused by large occlusion edges at the boundaries of textured regions. Contrast boundaries appear as blobs in the Laplacian of Gaussian image. To remove their effect, we use the Laplacian of Gaussian image at a slightly coarser scale. Blobs caused by the texture at the fine scale do not appear at this coarser scale, while the contrast boundaries, as well as all other blobs at coarser scales, remain. This coarse blob image filters the fine blobs - blobs at the coarser scale are removed from the fine scale image. Then, summation, whether with a simple scan operation, or Gaussian filtering, can determine the blob density at the fine scale only. This is one example where multiple spatial scales are used in the present implementation of the Vision Machine.

5. The Integration Stage and MRF

Whereas it is reasonable that combining the evidence provided by multiple cues - for example, edge detection, stereo and color - should provide a more reliable map of the surfaces than any single cue alone, it is not obvious how this integration can be accomplished. The various physical processes that contribute to image formation - ***surface depth, surface orientation, albedo (Lambertian and specular component), illumination*** - are coupled to the image data, and therefore to each other, through the imaging equation. The coupling is, however, difficult to exploit in a robust way, since it depends critically on the reflectance and imaging models. We argue that the coupling of the image data to the surface and illumination properties is of a more qualitative and robust sort at locations in which image brightness changes sharply and surface properties are discontinuous, in short, at edges. The intuitive reason for this is that at discontinuities, the coupling between different physical processes and the image data is robust and qualitative. For instance, a depth discontinuity usually originates a brightness edge in the image, and a motion boundary often corresponds to a depth discontinuity (and an brightness edge) in the image. This view suggests the following integration scheme for restoring the data provided by early modules. The results provided by stereo, motion and other visual cues are typically noisy and sparse. We can improve them by exploiting the fact that they should be smooth, or even piecewise constant (as in the case of the albedo), between discontinuities. We can exploit *a priori* information about generic properties of the discontinuities themselves: for instance, that they usually are continuous and non intersecting.

The idea is then to detect discontinuities in each cue, say depth, simultaneously with the approximation of the depth data. The detection of discontinuities is helped by information on the presence and type of discontinuities in the surfaces and surface properties (see Figure 1), which are coupled to the brightness edges in the image.

Notice that reliable detection of discontinuities is critical for a vision system, since discontinuities are often the most important locations in a scene: depth discontinuities, for example, normally correspond to the boundaries of an object or an object part. The idea is thus to couple different cues through their discontinuities and to use information from several cues simultaneously to help refine the initial estimation of discontinuities, which are typically noisy and sparse.

How can this be done? We have chosen to use the machinery of Markov Random Fields (MRFs), initially suggested for image processing by Geman and Geman [1984]. In the following we will give a brief, informal outline of the technique and of our integration scheme. More detailed information about MRFs can be found in Geman and Geman [1984] and Marroquin et.al. [1987]. Gamble and Poggio [1987] describe an earlier version of our integration scheme and its implementation, outlined in the next section.

5.1. MRF Models

Consider the prototypical problem of approximating a surface given sparse and noisy data (depth data), on a regular 2D lattice of sites. We first define the *prior probability of the class of surfaces we are interested* in. The probability of a certain depth at any given site in the lattice depends only upon neighboring sites (the Markov property). Because of the Clifford-Hammersley theorem, the prior probability is guaranteed to have the Gibbs form

$$P(f) = \frac{1}{Z} e^{-\frac{U(f)}{T}} \tag{7}$$

where Z is a normalization constant, T is called temperature, and $U(f) = \sum_C U_C(f)$ is an energy function that can be computed as the sum of local contributions from each neighborhood. The sum of the ***potentials***, $U_C(X)$, is over the neighborhood's ***cliques***. A clique is either a single lattice site or a set of lattice sites such that any two sites belonging to it are neighbors of one another. Thus $U(f)$ can be considered as the sum over the possible configurations of each neighborhood [see Marroquin et.al., 1987]. As a simple example, when the surfaces are expected to be smooth, the prior probability can be given in terms of

$$U_c(f) = (f_i - f_j)^2 \tag{8}$$

where i and j are neighboring sites (belonging to the same clique).

If a model of the observation process is available (i.e., a model of the noise), then one can write the conditional probability $P(g/f)$ of the sparse observation g for any given surface f. Bayes Theorem then allows one to write the posterior distribution

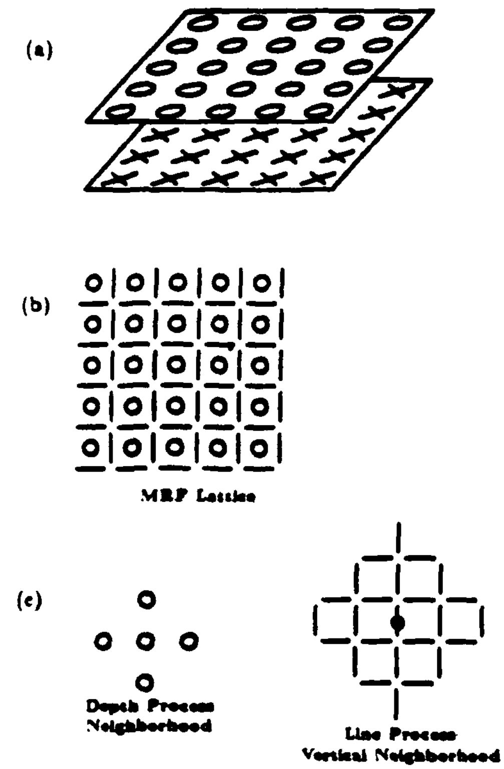


Figure 5: The MRF lattice

$$P(f/g) = \frac{1}{Z} e^{\frac{-U(f/g)}{T}} \qquad (9)$$

In the simple earlier example, we have (for Gaussian noise)

$$U(f/g) = \sum_{C} \alpha\gamma_i (f_i - g_i)^2 + (f_i - f_j)^2 \qquad (10)$$

where $\gamma_i = 1$ only where data are available. More complicated cases can be handled in a similar manner.

The posterior distribution cannot be solved analytically, but sample distributions with the probability distribution of Equation (3) can be obtained using Monte Carlo techniques such as the Metropolis algorithm. These algorithms sample the space of possible surfaces according to the probability distribution $P(f/g)$ that is determined by the prior knowledge of the allowed class of surfaces, the model of noise, and the observed data. In our implementation, a highly parallel computer generates a sequence of surfaces from which, for instance, the surface corresponding to the maximum of $P(f/g)$ can be found. This corresponds to finding the global minimum of $U(f/g)$ (simulated annealing is one of the possible techniques). Other criteria can be used: Marroquin [1985] has shown that the average surface f under the posterior distribution is often a better estimate which can be obtained more efficiently simply by finding the average value of f at each lattice site.

One of the main attractions of MRFs is that the prior probability distribution can be made to embed more sophisticated assumptions about the world. Geman and Geman [1984] introduced the idea of another process, the line process, located on the dual lattice (see Figure 5), and representing explicitly the presence or absence of discontinuities that break the smoothness assumption (Equation (2)). The associated prior energy then becomes

$$U_C(f) = (f_i - f_j)^2(1 - l_i^j) + \beta V_C(l_i^j) \qquad (11)$$

where l is a binary line element between site i, j. V_C is a term that reflects the fact that certain configurations of the line process are more likely than others to occur. In our world, depth discontinuities are usually themselves continuous, non-intersecting, and rarely isolated joints. These properties of physical discontinuities can be enforced locally by defining an appropriate set of energy values $V_C(l)$ for different configurations of the line process in the neighborhood of the site (notice that the assignment of zero energy values to the non-central cliques mentioned in Gamble and Poggio [1987] is wrong, as pointed out to us by Tail Symchony).

5.2. Organization of Integration

It is possible to extend the energy function of Equation (5) to accommodate the interaction of more processes and of their discontinuities. In particular, we have extended the energy function to couple several of the early vision modules (depth, motion, texture and color) to brightness edges in the image. This is a central point in our integration scheme: brightness edges guide the computation of discontinuities in the physical properties of the surface, thereby coupling surface depth, surface orientation, motion, texture and color, each to the image brightness data and to each other. The reason for the role of brightness edges is that changes in surface properties usually produce large brightness gradients in the image. It is exactly for this reason that edge detection is so important in both artificial and biological vision.

The coupling to brightness edges may be done by replacing the term $V_C(l_i^j)$ in the last equation with the term

$$V(l, e) = g(e_i^j, V_C(l_i^j)) \qquad (12)$$

with e_i^j representing a measure of the presence of an brightness edge between site i, j. The term g has the effect of modifying the probability of the line process configuration depending on the brightness edge data ($V(l,e) = -log\ p(l/e)$). This term facilitates formation of discontinuities (that is, l_i^j) at the locations of brightness edges. Ideally, the brightness edges (and the neighboring image properties) activate, with different probabilities, the different surface discontinuities (see Figure 1) which in turn are coupled to the output of stereo, motion, color, texture, and possibly other early algorithms.

We have been using the MRF machinery with prior energies like that given in Equations (11) and (12) (see also Figure 1) to integrate edge brightness data with stereo, motion and texture information on the MIT Vision Machine System.

We should emphasize that our present implementation represents a subset of the possible interactions shown in Figure 1, itself only a simplified version of the organization of the likely integration process. The system will be improved in an incremental fashion, including pathways not shown in Figure 1, such as feed-backs from the results of integration into the matching stage of the stereo and motion algorithms.

5.3. Algorithms: Deterministic and Stochastic

A few disclaimers are in order here. We have chosen to use MRF models because of their generality and theoretical attractiveness. This does not imply that stochastic algorithms must be used. For instance, in the cases in which the MRF model reduces to standard regularization [Marroquin et.al., 1987] and the data are given on a regular grid, the MRF formulation leads not only to a purely deterministic algorithm, but also to a convolution filter.

We expect that during our research we will define deterministic algorithms that are either equivalent to a MRF formulation, or are a good approximation to the stochastic Monte Carlo algorithms. More specifically, we expect that the probabilistic formulation of MRF is in a sense too general, and therefore too inefficient. Remember that MRF models are quite general (for instance, regularization can be regarded from a probabilistic point of view as an instance of MRF).

6. Illustrative Results

Figures 7 and 8 show the results of the Vision Machine applied to the scene in Figure 6 and some of the intermediate steps. Figure 7 show the brightness edges computed by the Canny algorithm at two different spatial scales ($\sigma = 2.5$ and $\sigma = 4$). We show neither the stereo pair nor the motion sequence in which the teddy bear was rolling slightly on his back from one frame to the next. The results given by the stereo, motion, texture and color algorithms, after an initial smoothing to make them dense [see Gamble and Poggio, 1987], are shown in the first column on the left of Figure 8 (from top to bottom). They represent the input to the MRF machinery that integrates each of those data sets with the brightness edges. The color algorithm uses the edges at the coarser resolution, since we want to avoid detecting texture marks on the surface; the other cues are integrated with the Canny edges at a smaller scale ($\sigma = 2.5$). The central column of Figure 8 shows the reconstructed depth, color (the quantity H defined earlier), texture and motion flow; the left column show the discontinuities found by the MRF machinery in each of the cues. Processing of the stereo output finds depth discontinuities in the scene (mainly the outlines of the teddy, plus a fold of a wet suit protruding outward). Motion discontinuities are found by the MRF machinery with help from brightness edges. The color boundaries show regions of constant surface color, independently of its shading: notice, for instance, that brightness edges inside the teddy bear, due to shading, do not appear as color edges (the color images were taken from a different camera). The texture boundaries correspond quite well to different textured surfaces.

Figure 9 shows that the union of the discontinuities in depth and motion for the scene of Figure 6 gives a rather good "cartoon" of the original scene. At the same time, our integration algorithm achieves a preliminary classification of the brightness edges in the image, in terms of their physical origin. A more complete classification will be achieved by the full scheme of Figure 1: the lattices at the top classify the different types of discontinuities in the scene. The set of such discontinuities in the various physical processes should represent a good set of data for later recognition stages.

7. The Future

The Vision Machine is evolving rapidly. We plan to add other early vision algorithms (such as shape-from-shading) and to develop further the ones already

Figure 6: Grey-level image of a natural scene processed by the Vision Machine

Figure 7: Canny edges of the image in Figure 6

implemented (especially color and texture). Most of the future effort will be directed towards a more satisfactory integration: we will define and implement a scheme of the type shown in Figure 1. Finding the correct values of the parameter is critical for the practical success of the MRF technique; thus we will attempt to find useful solutions to the parameter estimation problem, an issue strictly related to learning from examples. An important step in the very near future will be the implementation of recognition algorithms operating on the output of the integration stage.

7.1. Towards Recognition

The output of the integration stage provides a set of edges labeled in terms of physical discontinuities of the surface properties. They should represent a good input to a model-based recognition algorithm like the ones described by Dan Huttenlocher and by Todd Cass in these Proceedings. In particular, we are interfacing the Vision Machine as implemented so far with the Cass algorithm. Initially, we will use only discontinuities for recognition; later we will use also the information provided by the MRFs on the surface properties *between* discontinuities. Notice that in the full system we may have several visual routines [see Poggio et.al., these Proceedings) operating also on the maps of physical discontinuities and performing task-dependent grouping operations before recognition.

7.2. Learning and Parameter Estimation

Using the MRF model involves an energy function which has several free parameters, in addition to the many possible neighborhood systems. The values of these parameters determine a distribution over the configuration-space to which the system converges, and the speed of convergence. Thus rigorous methods for estimating these parameters are essential for the practical success of the method and for meaningful results. In some cases, parameters can be learned from the data: e.g., texture parameters [Geman and Graffigne, 1987], or neighborhood parameters (for which a cellular automaton model may be the most convenient for the purpose of learning). There are general *statistical* methods which can be used for parameter estimation:

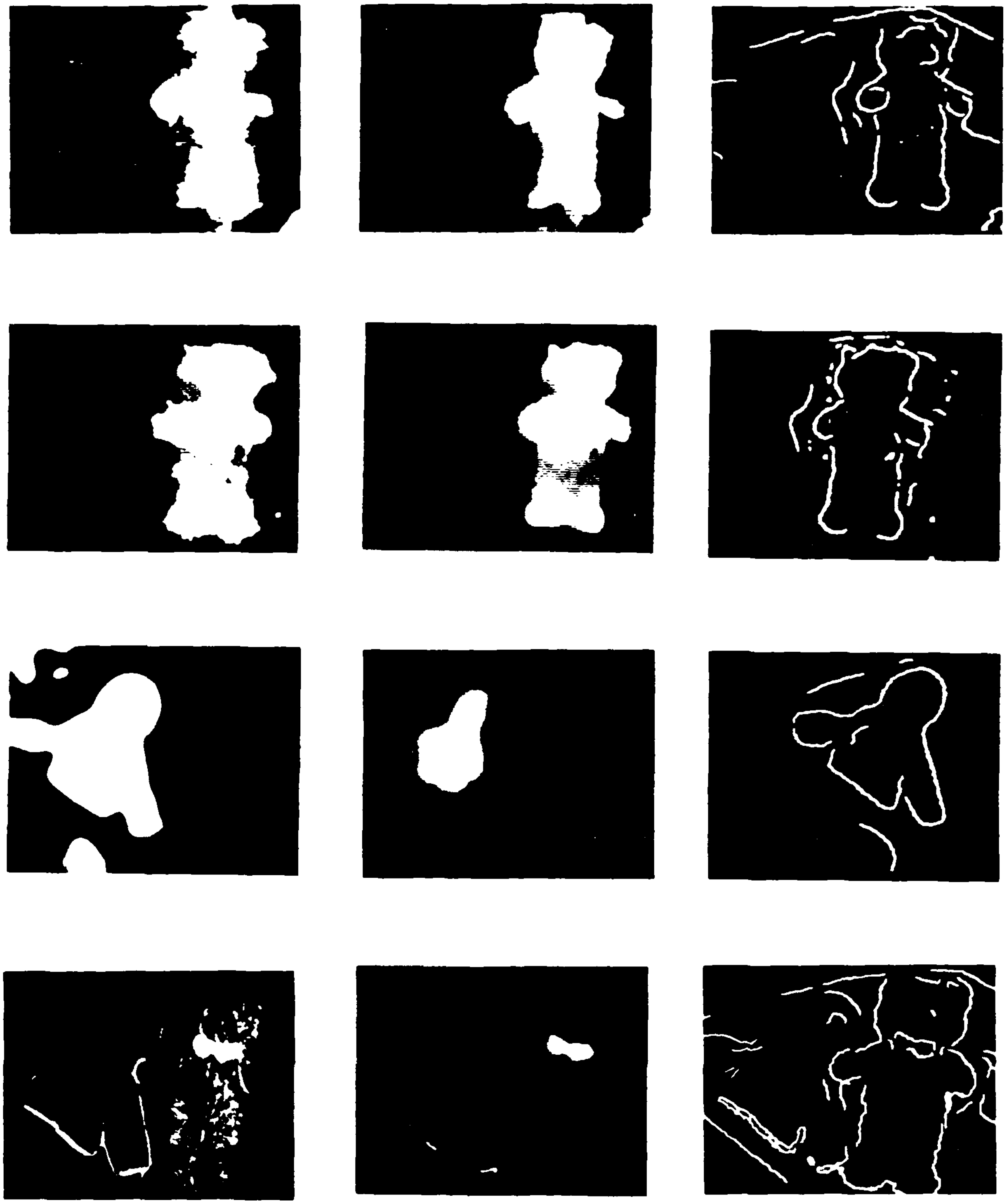


Figure 8: MRF results for stereo, motion, texture and color

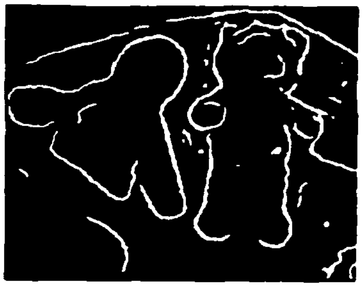


Figure 9: Union of depth and motion discontinuities

- A maximum likelihood estimate – one can use the indirect iterative EM algorithm [Dempster et.al., 1977], which is most useful for maximum likelihood estimation from incomplete data [see Marroquin, 1987 for a special case]. This algorithm involves the iterative maximization (over the parameter space) of the expected value of the likelihood function given that the parameters take the values of their estimation in the previous iteration. Alternatively, a search constrained by some statistics for a minimum of an appropriate merit function may be employed [see Marroquin, 1987].
- A smoothing (regularization) parameter can be estimated using the methods of cross-validation or unbiased risk, to minimize the mean square error. In cross-validation, an estimate is obtained omitting one data point. The goal is to minimise the distance between the predicted data point (from the estimate above with the point omitted) and the actual value, for all points.

In the case of Markov Random Fields, some more specific approaches are appropriate for parameter estimation:

1) Besag [1974] suggested conditional maximum likelihood estimation using coding methods, maximum likelihood estimation with unilateral approximations on the rectangular lattice, or "maximum pseudolikelihood" – a method to estimate parameters for homogeneous random fields [see Geman and Graffigne, 1987].

2) For the MPM estimator, where a fixed temperature is yet another parameter to be estimated, one can try to use the physics behind the model to find a temperature with as little disorder as possible and still reasonable time of convergence to equilibrium (e.g., away from "phase-transition").

An alternative asymptotic approach can be used with smoothing (regularization) terms: instead of estimating the smoothing parameter, let it tend to 0 as the temperature tends to 0, to reduce the smoothing close to the final configuration [see Geman and Geman, 1987].

In summary, we plan to explore three distinct stages for parameter estimation in the integration stage of the Vision Machine:

- *Modeling* (from the physics of surfaces, of the imaging process and of the class of scenes to be analyzed and the tasks to be performed) and the form of the prior and of some conditional probabilities involved (e.g., the type of physical edges from properties of the measurements, such as characteristics of the brightness data). Range of allowed parameter values may also be established at this stage (e.g., minimum and maximum brightness value in a scene, depth differences, positivity of certain measurements, distribution of expected velocities, reflectance properties, characteristics of the illuminant, etc.).
- *Estimating* of parameter values from set of examples in which data and desired solution are given. This is a *learning stage*. We may have to use days of CM time and, at least initially, synthetic images to do this.
- *Tuning* of some of the parameters directly from the data (by using EM algorithm, cross-validation, Besag's work, or various types of heuristics).

The dream is that at some point in the future the Vision Machine will run all the time, day and night, looking about and learning on its own to see better and better.

7.3. Fast Vision: The Role of Time Smoothness

The present version of the Vision Machine processes only isolated frames. Even our motion algorithm takes as input simply a sequence of two images. The reason for this is, of course, limitations in raw speed. We cannot perform all of the processing we do at video rate (say, 30 frames per second), though this goal is certainly within present technological capabilities. If we could process frames at video rate, we could exploit constraints in the time dimension similar to the ones we are already exploiting in the space domain. Surfaces – and even the brightness array itself – do not usually change too much from frame to frame. This is a constraint of smoothness in time, which is valid almost everywhere, but not *across* discontinuities in time. Thus one may use the same MRF technique, applied to the output of stereo, motion, color, and texture, and enforcing continuity in time (if there are no discontinuities), that is, exploiting the redundancy in the sequence of frames.

We believe that the surface reconstructed from a stereo pair usually does not need to be recomputed completely when the next stereo pair is taken a fraction of a second later. Of course, the role of the MRFs may be accomplished in this case by some more specific and more efficient deterministic method such as, for instance, a form of Kalman filtering. Notice that space-time MRFs applied to the brightness arrays would yield spatiotemporal interpolation and approximation of a kind already considered [Fahle and Poggio, 1980; Poggio, Nielsen and Nishihara, 1982; Bliss, 1985].

7.4. A VLSI Vision Machine?

Our Vision Machine is mostly specialized software running on a general purpose computer, the Connection Machine. This is a good system for the present stage of experimentation and development. Later, once we have perfected and tested the algorithms and the overall system, it will make *sense* to compile the software in silicon in order to produce a faster, cheaper, and smaller Vision Machine. We are presently planning to use VLSI technologies to develop some initial chips as a first step toward this goal. In this section, we will outline some thoughts about VLSI implementation of the Vision Machine.

Algorithms and Hardware

We realize that our specialized software vision algorithms are not, in general, optimized for hardware implementation. So, rather than directly "hardwiring algorithms" into standard computing circuitry, we will be investigating "algorithmic hardware" designs that utilize the local, symmetric nature of early vision problems. This will be an iterative process, as the algorithm influences the hardware design and as hardware constraints modify the algorithm.

Degree of Parallelism

Typical vision tasks require tremendous amounts of computing power and are usually parallel in nature. As an example, biology uses highly parallel networks of relatively slow components to achieve sophisticated vision systems. However, when implementing our algorithms into silicon integrated circuits, it is not clear what level of parallelism is necessary. While biology is able to use three dimensions to construct highly interconnected parallel networks, VLSI is limited to $2\frac{1}{2}$ dimensions, making highly parallel networks much more difficult and costly to implement. However, the electrical components of silicon integrated circuits are approximately four orders of magnitude faster than the electrochemical components of biology. This suggests that pipelined processing or other methods of time-sharing computing power may be able to compensate for the lower degree of connectivity of silicon VLSI. Clearly, the architecture of a VLSI vision system may not resemble any biological vision systems.

Signal Representation

Within the integrated circuit, the image data may be represented as a digital word or an analog value. While the advantages of digital computation are its accuracy and speed, digital circuits do not have as high functionality per device as analog circuits. Therefore, analog circuits should allow much denser computing networks. This is particularly important for the integration of computational circuitry and photosensors, which will help to alleviate the I/O bottleneck typically experienced whenever image data are serially transferred between Vision Machine components. However, analog circuits are limited in accuracy and are difficult to characterize and design.

The primary motivation for a VLSI implementation of our Vision Machine is to increase the computational speed and reduce the physical size of the components with the eventual goal of real-time, mobile vision systems. While the main computational engine of our Vision Machine is the Connection Machine, which is a very powerful and flexible SIMD computer, specific VLSI implementations will attempt to tradeoff computational flexibility for faster performance and higher degree of integration. A VLSI implementation of our Vision Machine can offer significant improvements in performance that would be difficult or impossible to attain by other methods. Presently, we are specifically investigating the integration of charge coupled devices for photosensing and simple parallel computations, such as binomial convolution and patchwise correlation.

Legends

Figure 1: A diagram of the overall organization of the integration stage [see Gamble and Poggio, 1987 for a complementary diagram]. The output of each of the early visual cues (or algorithms) – stereo, motion, texture and color – are coupled to their own line process (the crosses), i.e., their discontinuities. They are also coupled to the discontinuities in the surface properties – occluding edges (both extremal

here, is that changes in surface properties usually produce large brightness gradients in the image.

The coupling to high brightness gradients may be done by replacing the term $V_C(l_i^j)$ in the last equation with the term:

$$V(l,b) = g(b_i^j, l_i^j) \qquad (6)$$

with b_i^j representing a measure of the strength of the brightness gradient (that is, of a brightness edge) between site i and j. The term g has the effect of modifying the probability of the line process configuration depending on the brightness edge data [for instance, $g(b_i^j, l_i^j) = b_i^j(1 - l_i^j)$]. This term facilitates formation of discontinuities (that is, $l_i^j = 1$) at the locations of sharp brightness changes, without restricting them only to brightness edges. High values of the brightness gradient (together with image data in the neighborhood) activate with different probabilities the different types of surface discontinuities (see Fig. 1) which, in turn, are coupled to the output of stereo, motion, color, texture, and possibly other early vision algorithms.

We have been using the MRF machinery with prior energies like that given in Eqs. 5 and 6 (see also Fig. 1) to integrate edge brightness data with stereo, motion, color, and texture information on the MIT Vision Machine System. The system consists of a two-camera eye-head input device and a 16K Connection Machine. All the early vision algorithms—edge detection, stereo, motion, color, and texture—as well as the MRF algorithm, now run on the Connection Machine several hundred times faster than on a conventional machine. The results of integrating brightness edges with a parallel stereo algorithm (*20*) are shown in Fig. 3. In a similar way, the optical flow and its boundary from the same scene are computed from motion data (*21*) and brightness edges (*5, 6, 22, 23*). Simple examples of a similar integration performed with texture and color data are shown in Fig. 3, d and e. The texture algorithm is a greatly simplified parallel version of the texture algorithm developed by Voorhees and Poggio (*24*). It measures the level density of "blobs" extracted from the image through a filtering process involving center-surround filters with appropriate size and threshold. The color algorithm provides a local measure of hue, $H = R/(R + G)$, where R and G are the measurements in the red and green channels, respectively, of a digital color camera. Under certain conditions [A. C. Hurlbert, see (*25*)], this ratio is independent of illumination and three-dimensional (3-D) shape. An MRF model that enforces local constancy of the hue H uses these dense but noisy data to segment the image into regions of different constant reflectance (*26*). The coupling with brightness edges facilitates finding the boundaries: usually sharp changes in the ratio H correspond to a subset of the brightness edges.

The union of the discontinuities in depth, motion, and texture for the scene of Fig. 3 gives a "cartoon" of the original scene. Notice that this "cartoon" represents discontinuities in the physical properties of 3-D surfaces that are well defined, whereas brightness "discontinuities" are not well defined in terms of surface properties. Our integration algorithm achieves a preliminary classification of the edges in the image, in terms of their physical origin. A more complete classification may be achieved by implementing the full scheme of Fig. 1; the lattices at the top classify the different types of discontinuities in the scene: depth discontinuities, orientation discontinuities, albedo edges, specular edges, and shadow edges. The set of such discontinuities in the various physical processes seems to represent a good set of data for later recognition. In some preliminary experiments we have successfully used a parallel, model-based recognition system (*27*) on the discontinuities (stereo and motion) provided by our MRF scheme (*28*).

Our present implementation represents a subset of the possible interactions shown in Fig. 1, itself only a simplified version of the organization of the likely integration process. As described elsewhere (*5, 26*), the system will be improved in an incremental fashion, including pathways not shown in Fig. 1, such as feedback from the results of integration into the matching stage of the stereo and motion algorithms.

The highly parallel algorithms we have described (*29*) map quite naturally onto an architecture such as the Connection Machine, which consists of 64K simple one-bit processors with local and global connection capabilities. The same algorithms also map onto very large scale integration (VLSI) architectures of fully analog elements (we have successfully experimented with a version of Eqs. 5 and 6, in which l is a continuous variable), mixed analog and digital components and purely digital processors (similar to a much simplified and specialized Connection Machine).

A plausible organization of visual integration as sketched in Fig. 1 may be found ultimately by theory and by computer experiments of the type described here. We believe that psychophysical and physiological data about the integration stage in the mammalian visual system may be helpful in guiding our theoretical and computational work. The system described here has already triggered a series of psychophysical experiments in order to establish whether and how brightness edges aid human computation of surface discontinuities (*30*).

REFERENCES AND NOTES

1. T. Poggio and V. Torre, *A.I. Memo No. 773, C.B.I.P. Paper No. 001* (Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, 1984).
2. M. Bertero, T. Poggio, V. Torre, *A.I. Memo No. 924* (Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, 1987); also *Proc. IEEE*, in press.
3. T. Poggio, V. Torre, C. Koch, *Nature* **317**, 314 (1985).
4. T. Poggio, *Working Paper No. 285* (Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, 1985).
5. E. B. Gamble and T. Poggio, *A.I. Memo No. 970* (Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, 1987).
6. J. Hutchinson, C. Koch, J. Luo, C. Mead, *IEEE Computer Magazine* **21**, 52 (March 1988).
7. T. Poggio *et al.*, in *Proceedings: Image Understanding Workshop*, Los Angeles, February 1987 (Morgan Kaufmann, San Mateo, CA, 1987), pp. 41–54.
8. P. B. Chou and C. M. Brown, in *Proceedings: Image Understanding Workshop*, Los Angeles, February 1987 (Morgan Kaufmann, San Mateo, CA, 1987), pp. 663–670.
9. P. B. Chou and C. M. Brown, in *Proceedings: International Joint Conference on Artificial Intelligence*, Milan, August 1987 (Morgan Kaufmann, San Mateo, CA, 1987), pp. 779–782.
10. P. B. Chou and C. M. Brown, in *Proceedings: Image Understanding Workshop*, Cambridge, April 1988, (Morgan Kaufmann, San Mateo, CA, 1988), pp. 214–221.
11. H. G. Barrow and J. M. Tenenbaum, in *Computer Vision Systems*, A. R. Hanson and E. M. Riseman, Eds. (Academic Press, New York, 1978), pp. 3–26.
12. S. Geman and D. Geman, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-6**, 721 (1984).
13. W. Hoff and N. Ahuja, in *Proceedings of the International Conference on Computer Vision*, London, June 1987 (IEEE, Washington, DC, 1987), pp. 284–294.
14. A. Blake and A. Zisserman, *Visual Reconstruction* (MIT Press, Cambridge, MA, 1987).
15. J. Aloimonos and C. M. Brown, in *Advances in Computer Vision*, C. Brown, Ed. (Erlbaum, Hillsdale, NJ, 1987), pp. 115–163.
16. F. S. Cohen and D. B. Cooper, in *Proceedings of SPIE Conference on Advances in Intelligent Robotics Systems*, Cambridge, MA, November 1983 (SPIE, the International Society for Optical Engineering, Bellingham, WA, 1983).
17. J. L. Marroquin, S. Mitter, T. Poggio, in *Proceedings: Image Understanding Workshop*, L. Baumann, Ed., Miami Beach, FL, December 1985 (Scientific Applications International Corporation, San Diego, CA, 1985), pp. 293–309.
18. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, *J. Phys. Chem.* **21**, 1087 (1953).
19. J. L. Marroquin, *Probabilistic Solution of Inverse Problems*, thesis, Massachusetts Institute of Technology (1985).
20. M. Drumheller and T. Poggio, in *Proceedings of IEEE Conference on Robotics and Automation* (IEEE, Washington, DC, 1986), pp. 1439–1448.
21. J. J. Little, H. H. Bülthoff, T. Poggio, in *Proceedings: Image Understanding Workshop*, Los Angeles, February 1987 (Morgan Kaufmann, San Mateo, CA, 1987), pp. 915–920.
22. D. W. Murray and B. F. Buxton, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-9** (no. 2), 220 (1987).
23. A. L. Yuille, *A.I. Memo No. 987* (Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge 1987).
24. H. Voorhees and T. Poggio, *Nature* **333**, 364 (1988).
25. T. Poggio *et al.*, in *Proceedings: Image Understanding Workshop*, L. Baumann, Ed., Miami Beach, FL, December 1985 (Scientific Applications Interna-

tional Corporation, San Diego, CA, 1985), pp. 25–39

26. T. Poggio *et al.*, in *Proceedings Image Understanding Workshop*, Cambridge, April 1988 (Morgan Kaufmann, San Mateo, CA, 1988), pp. 1–12.

27. T. A. Cass, in *ibid.*, pp. 640–650

28. We have exploited the labeling of discontinuities (E. B. Gamble, D. Geiger, T. Poggio, D. Weinshall, in preparation) in recognition experiments. In addition, our integration scheme allows us to segment the scene into different depth planes, for instance, thereby considerably reducing the combinatorics of model-based recognition.

29. Our formulation of the integration problem in terms of MRF does not imply that the algorithms are necessarily stochastic. Deterministic approximations to the more general stochastic schemes may work quite well, especially in situations where redundant and contradictory data from several sources effectively set the initial state of the system close to the solution. We have, in fact, found that gradient descent in the space of the depth and the line process often works quite well. We routinely use a mixed deterministic and stochastic strategy (*17*) in which the continuous (depth) process is deterministically updated while the line process is updated stochastically. Other strategies may also be effective (*8*), such as space-variant filtering, for instance, coupled with edge detection. In addition, time-dependent schedules of the coupling parameters can be useful. They are somewhat similar to simulated annealing, which can also be effectively used, though it is quite slow.

30. H. H. Bulthoff, personal communication.

31. J. F. Canny, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-8** (no. 6), 679 (1986).

32. This report describes research done within the Artificial Intelligence Laboratory. Support for the A. I. Laboratory's artificial intelligence research is provided by the Advanced Research Projects Agency of the Department of Defense under Army contract DACA76-85-C-0010 and in part under Office of Naval Research (ONR) contract N00014-85-K-0124. Support for this research is also provided by a grant from ONR, Engineering Psychology Division, and by a gift from the Artificial Intelligence Center of Hughes Aircraft Corporation to T. Poggio.

4 May 1988; accepted 12 August 1988

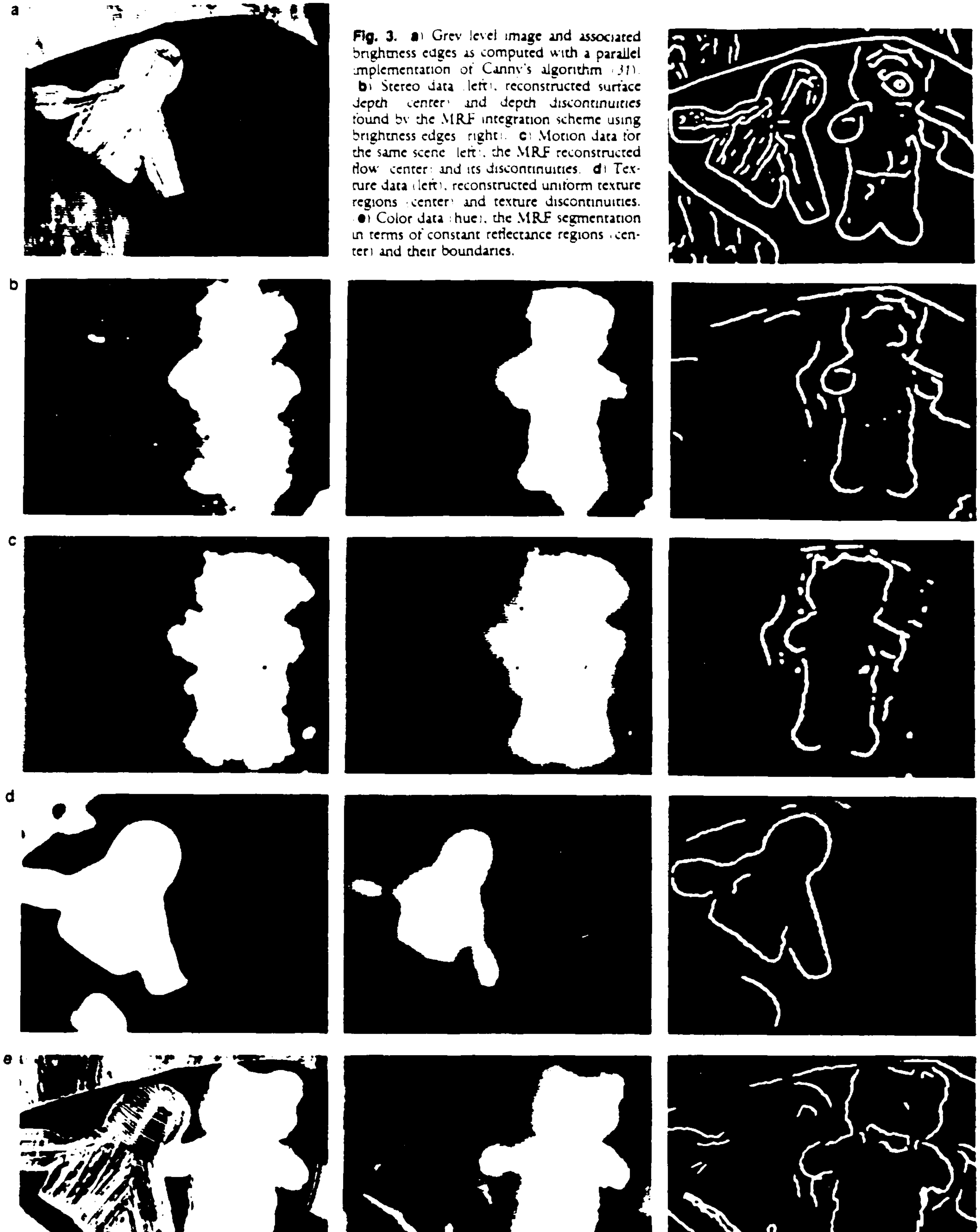


Fig. 3. (**a**) Grey level image and associated brightness edges as computed with a parallel implementation of Canny's algorithm (31). (**b**) Stereo data (left), reconstructed surface depth (center) and depth discontinuities found by the MRF integration scheme using brightness edges (right). (**c**) Motion data for the same scene (left), the MRF reconstructed flow (center) and its discontinuities. (**d**) Texture data (left), reconstructed uniform texture regions (center) and texture discontinuities. (**e**) Color data (hue), the MRF segmentation in terms of constant reflectance regions (center) and their boundaries.

quantity analogous to temperature in statistical mechanics, and $U(f) = \Sigma U_i(f)$ is an energy function that can be computed as the sum of local contributions from each lattice site i. The energy at each lattice site $U_i(f)$ is, itself, a sum of the potentials, $U_C(f)$, of each site's cliques. A clique is either a single lattice site or a set of lattice sites such that any two sites belonging to it are neighbors of one another (5, 17). As a simple example, when the surfaces are expected to be smooth (like a membrane), the prior energy can be given in terms of

$$U_i(f) = \sum (f_i - f_j)^2 \qquad (2)$$

where j is a neighboring site to i (that is, i and j belong to the same clique).

If a model of the observation process is available (that is, a model of the noise), then one can write the conditional probability $P(g/f)$ of the sparse observation g for any given surface f. Bayes's theorem then allows one to write the posterior distribution:

$$P(f/g) = \frac{1}{Z}e^{-U(f/g)/T} \qquad (3)$$

In the example of Eq. 2, we have (for Gaussian noise):

$$U_i(f/g) = \sum (f_i - f_j)^2 + \alpha\gamma_i(f_i - g_i)^2 \qquad (4)$$

where $\gamma_i = 1$ only where data are available, and otherwise $\gamma_i = 0$. More complicated cases can be handled in a similar manner (5).

The maximum of the posterior distribution or other related ^estimates cannot be computed analytically, but sample distributions with the probability distribution of Eq. 3 can be obtained by means of Monte Carlo techniques such as the Metropolis algorithm (18). These algorithms sample the space of possible surfaces according to the probability distribution $P(f/g)$ that is determined by the prior knowledge of the allowed class of surfaces, the model of noise, and the observed data. In our implementation, a highly parallel computer generates a sequence of surfaces from which, for instance, the surface corresponding to the maximum of $P(f/g)$ can be found. This corresponds to finding the global minimum of $U(f/g)$ (simulated annealing is one of the possible techniques). Other criteria can be used: Marroquin (19) has shown that the average surface f under the posterior distribution is often a better estimate, which can be obtained more efficiently simply by finding the average value of f at each lattice site.

One of the main attractions of MRF models is that the prior probability distribution can be made to embed more sophisticated assumptions about the world. Geman and Geman (12) introduced the idea of another process, the line process, located on the dual lattice (see Fig. 2), and representing explicitly the presence or absence of discontinuities that break the smoothness assumption (Eq. 2). The associated prior energy then becomes:

$$U_i(f,l) = \sum_j (f_i - f_j)^2(1 - l_i^j) + \beta V_C(l_i^j) \qquad (5)$$

where l_i^j is a binary line element between site i and j. The term $V_C(l_i^j)$ reflects the fact that certain configurations of the line process are more likely than others to occur. Depth discontinuities are usually themselves continuous, nonintersecting, and rarely isolated points. These properties of physical discontinuities can be enforced locally by defining an appropriate set of energy values $V_C(l_i^j)$ for different configurations of the line process (5, 12, 17).

It is possible to extend the energy function of Eq. 5 to accommodate the interaction of more processes and of their discontinuities. In particular, we have extended the energy function to couple several of the early vision modules (depth, motion, texture, and color) to sharp changes of brightness in the image. This is a central point in our integration scheme: here we assume that changes of brightness guide the computation of discontinuities in the physical properties of the surface, thereby coupling surface depth, surface orientation, motion, texture, and color each to the image brightness data and to each other. The reason for the primary role of the gradient of brightness, as conjectured

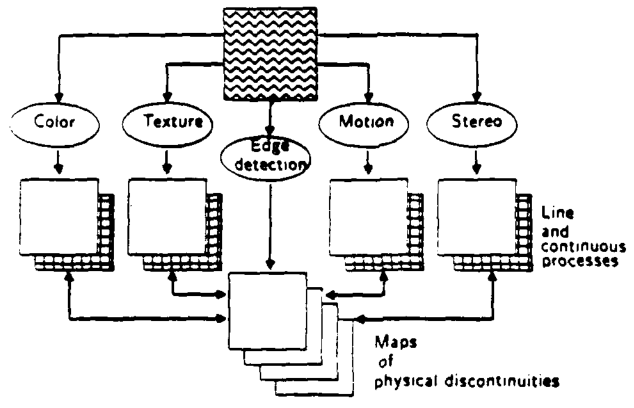

Color
Texture
Edge detection
Motion
Stereo
Line and continuous processes
Maps of physical discontinuities

Fig. 1. A sketch of the overall organization of the integration stage (5, 26). The outputs of the early visual cues (or algorithms)—stereo, motion, texture, and color—are coupled to their own line process (the crosses), that is, their discontinuities. They are also coupled to the discontinuities in the surface properties—occluding edges (both extremal edges and blades), orientation discontinuities, specular edges, texture marks (including albedo discontinuities), and shadow edges. The image data, especially the sharp changes in brightness labeled here as edges, are input to the lattices that represent the discontinuities in the physical properties of the surfaces. The brightness edges may be completed before integration (in some cases this may lead to "subjective contours") by the equivalent of a higher order MRF that reflects long-range constraints of colinearity and continuation and even hypotheses from the recognition stage, which is then expected to use the set of discontinuities at the top as its main input. Our present implementation does not couple the different types of physical discontinuities: sharp changes in brightness are directly coupled to the line processes of each of the cues. The individual modules are therefore integrated with each other only indirectly, through the brightness edges.

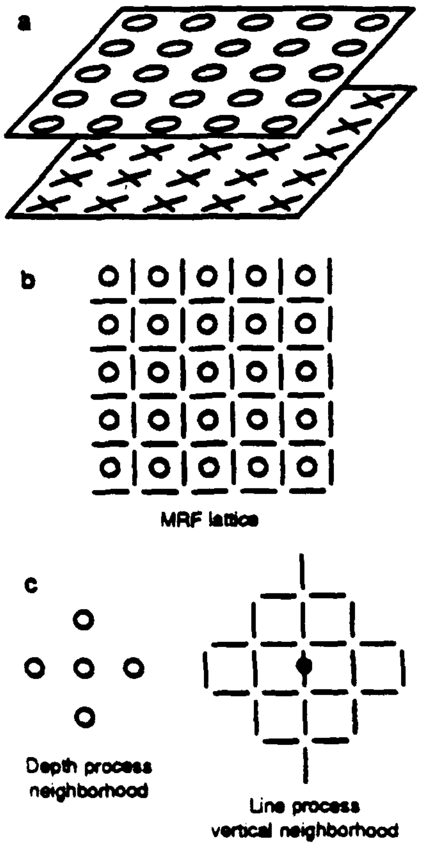

a
b
MRF lattice
c
Depth process neighborhood
Line process vertical neighborhood

Fig. 2. (**a**) Coupled MRF lattices: the circles represent the continuous process (depth, motion, color, or texture) and the crosses [the lines in (**b**)] represent the associated line process, that is, the discontinuities. The neighborhoods of the continuous process and of the line process are shown in (**c**). The cost of an isolated line process is much higher than that of a continuous line.

edges and blades), orientation discontinuities, specular edges, texture marks (including albedo discontinuities), shadow edges. The image data – mainly its intensity edges – are input to the lattices that represent the discontinuities in the physical properties of the scene. Our present implementation does not distinguish the different types of physical discontinuities: intensity edges are directly coupled to the line processes of each of the cues. The intensity edges can be completed and extended by the equivalent of a higher order MRF that reflects constraints of colinearity and continuation, and even hypotheses from the recognition stage, which uses the set of discontinuities at the top as its main input.

Figure 2: The Eye-Head System. See text.

Figure 3: See text.

Figure 4: See text.

Figure 5: A MRF lattice, consisting of depth (or color or texture) process elements (circles) with vertical and horizontal line elements (lines). The neighborhood of the depth process and of the line process are also shown, together with three of the several configurations of the line process. The cost of an isolated line process is much higher than that of a continuous line.

Figure 6: Grey-level image of a natural scene processed by the Vision Machine.

Figure 7: Canny edges of the image in Figure 6. See text.

Figure 8: See text.

Figure 9: See text.

Reading List

Barrow, H.G. and J.M. Tenenbaum. "Recovering Intrinsic Scene Characteristics from Images," In: **Computer Vision Systems**, A. Hanson and E. Riseman (eds.), Academic Press, New York, 1978.

Bertero, M., T. Poggio and V. Torre. "Ill-Posed Problems in Early Vision," *Artificial Intelligence Laboratory Memo 924*, Massachusetts Institute of Technology, Cambridge, MA, 1986.

Besag, J. "Spatial Interaction and the Statistical Analysis of Lattice systems," *J. Roy. Stat. Soc.*, **B34**, 75-83, 1972.

Blake, A. "On the Geometric Information Obtainable from Simultaneous Observation of Stereo Contour and Shading," *Technical Report CSR-205-86*, Dept. of Computer Science, University of Edinburgh, 1986.

Blelloch, G.E. "Scans as Primitive Parallel Operations," *Proc. Intl. Conf. on Parallel Processing*, 355-362, 1987.

Bliss, J. "Velocity Tuned Spatio-Temporal Interpolation and Approximation in Vision," Master's Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.

Brooks, R. "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, **RA-2**, 14-23, 1987.

Bülthoff H. and H. Mallot. "Interaction of Different Modules in Depth Perception," *Proc. First Intl. Conf. on Computer Vision*, Computer Society of the IEEE, Washington, DC, 295-305, 1987.

Bülthoff, H. and H. Mallot. "Interaction of Different Modules in Depth Perception: Stereo and Shading," *Artificial Intelligence Laboratory Memo 965*, Massachusetts Institute of Technology, Cambridge, MA, 1987.

Canny, J.F. "Finding Edges and Lines," *Artificial Intelligence Laboratory Technical Report 720*, Massachusetts Institute of Technology, Cambridge, MA, 1983.

Cornog, K.H. "Smooth Pursuit and Fixation for Robot Vision," Master's Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.

Dempster, A.P., N.M. Laird and D.B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm," *J. Roy. Stat. Soc.*, **B39**, 1-38, 1977.

Drumheller, M. and T. Poggio. "On Parallel Stereo," *Proc. Intl. Conf. on Robotics and Automation*, IEEE, 1986.

Fahle, M. and T. Poggio. "Visual Hyperacuity: Spatiotemporal Interpolation in Human Vision," *Proc. Roy. Soc. Lond. B*, **213**, 451-477, 1980.

Geman, D. and S. Geman. "Relaxation and Annealing with Constraints," *Complex Systems Technical Report 35*, Division of Applied Mathematics, Brown University, Providence, RI, 1987.

Geman, S. and D. Geman. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **6**, 1984.

Geman, S. and C. Graffigne. "Markov Random Field Image Models and their Applications to Computer Vision," *Proc. Intl. Congress of Mathematicians*, preprint, A.M. Gleason (ed.), 1987.

Gamble, E. and T. Poggio. "Integration of Intensity Edges with Stereo and Motion," *Artificial Intelligence Laboratory Memo 970*, Massachusetts Institute of Technology, Cambridge, MA, 1987.

Grimson, W.E.L. **From Images to Surfaces**, The MIT Press, Cambridge, MA, 1981.

Grimson, W.E.L. "A Computational Theory of Visual Surface Interpolation," *Phil. Trans. Roy. Soc. Lond. B*, **298**, 395-427, 1982.

Grimson, W.E.L. "Binocular Shading and Visual Surface Reconstruction," *Computer Vision, Graphics and Image Processing*, **28**, 19-43, 1984.

Hildreth, E.C. **The Measurement of Visual Motion**, The MIT Press, Cambridge, MA, 1983.

Hillis, D. "The Connection Machine," Ph.D. Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.

Horn, B.K.P. **Robot Vision**, The MIT Press, Cambridge, MA, 1986.

Hurlbert, A. "The Computation of Color Vision," Ph.D. Thesis, Dept. of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA, expected 1988.

Hurlbert, A. and T. Poggio. "Do Computers Need Attention?" *Nature*, **321**, 12, 1986.

Hurlbert, A. and T. Poggio. "Learning a Color Algorithm from Examples," *Artificial Intelligence Laboratory Memo 909/Center for Biological Information Processing Paper 25*, Massachusetts Institute of Technology, Cambridge, MA, 1987.

Huttenlocher, D. and S. Ullman. "Recognizing Rigid Objects by Aligning them with an Image," *Artificial Intelligence Laboratory Memo 937*, Massachusetts Institute of Technology, 1987.

Ikeuchi, K., and B.K.P. Horn. "Numerical Shape from Shading and Occluding Boundaries" *Artificial Intelligence*, **17**, 141-184, 1981.

Kender, J.R. "Shape from Texture: An Aggregation Transform that Maps a Class of Textures into Surface Orientation," *Proc. Sixth Intl. Joint Conf. on Artificial Intelligence*, Tokyo, 1979.

Kirkpatrick, S., C.D. Gelatt, Jr. and M.P. Vecchi. "Optimization by Simulated Annealing," *Science*, **220**, 1983.

Kruskal, C.P., L. Rudolph and M. Snir. "The Power of Parallel Prefix," *Proc. Intl. Conf. on Parallel Processing*, 180-185, 1985.

Lim, W. "Fast Algorithms for Labelling Connected Components in 2D Arrays," *Thinking Machines Corp. Technical Report NA86-1*, Cambridge, MA, 1986.

Little, J., Blelloch, G.E. and T. Cass. "Parallel Algorithms for Computer Vision on the Connection Machine," *Proceedings Intl. Conf. on Computer Vision*, 587-591, Los Angeles, 1987.

Little, J., Bülthoff, H. and Poggio, T. "Parallel Optical Flow Computation," *Proc. Image Understanding Workshop*, L. Bauman (ed.), Science Applications International Corp., McLean, VA, 915-920, 1987.

Little, J., Bülthoff, H. and Poggio, T. "Parallel Optical Flow Using Winner-Take-All Scheme," in preparation.

Mahoney, J.V. "Image Chunking: Defining Spatial Building Blocks for Scene Analysis," Master Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1987. Published as *Artificial Intelligence Laboratory Technical Report 980*, 1987.

Marr, D. **Vision**, Freeman, San Francisco, 1982.

Marr, D. and E. Hildreth. "Theory of Edge Detection," *Proc. Roy. Soc. Lond. B*, **207**, 187-217, 1980.

Marr, D. and T. Poggio. "Cooperative Computation of Stereo Disparity," *Science*, **194**, 283-287, 1976.

Marr, D. and T. Poggio, "A Computational Theory of Human Stereo Vision," *Proc. Roy. Soc. Lond. B*, **204**, 301-328, 1979.

Marroquin, J.L. "Deterministic Bayesian Estimation of Markov Random Fields with Applications to Computational Vision," *Proc. First Intl. Conf. on*

Computer Vision, Computer Society of the IEEE, Washington, DC, 1987.

Marroquin, J.L. "Probabilistic Solutions of Inverse Problems," *Artificial Intelligence Laboratory Technical Report 860*, Massachusetts Institute of Technology, Cambridge, MA, 1985.

Marroquin, J.L. "Surface Reconstruction Preserving Discontinuities," *Artificial Intelligence Laboratory Memo 792*, Massachusetts Institute of Technology, Cambridge, MA, 1984.

Marroquin, J.L., Mitter S. and T. Poggio. "Probabilistic Solution of Ill-Posed Problems in Computational Vision," *Proc. Image Understanding Workshop*, L. Bauman (ed.), Scientific Applications International Corp., McLean, VA, 1986. A more complete version appears in *J. Amer. Stat. Assoc.*, **82**, 76-89, 1987.

Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller. "Equation of State Calculations by Fast Computing Machines," *J. Phys. Chem.*, **21**, 1953.

Nishihara, H.K. "PRISM: A Practical Real-Time Imaging Stereo Matcher," *Artificial Intelligence Laboratory Memo 780*, Massachusetts Institute of Technology, Cambridge, MA, 1984.

Nishihara, H.K. and P.A. Crossley, "Measuring Photolithographic Overlay Accuracy and Critical Dimensions by Correlating inarized Laplacian of Gaussian Convolutions," *IEEE Trans. Pattern Matching and Machine Intell.*, **10**, 1988.

Poggio, T., K.R.K. Nielsen and H.K. Nishihara. "Zero-Crossings and Spatiotemporal Interpolation in Vision: Aliasing and Electrical Coupling Between Sensors," *Artificial Intelligence Laboratory Memo 675*, Massachusetts Institute of Technology, Cambridge, MA, 1982.

Poggio, G. and T. Poggio. "The analysis of stereopsis," *Ann. Rev. Neurosci.*, **7**, 379-412, 1984.

Poggio, T. "Early Vision: From Computational Structure to Algorithms and Parallel Hardware," *Computer Vision, Graphics, and Image Processing*, **31**, 1985.

Poggio, T. "Integrating Vision Modules with Coupled MRFs," *Artificial Intelligence Laboratory Working Paper 285*, Massachusetts Institute of Technology, Cambridge, MA, 1985.

Poggio, T. and staff. "MIT Progress in Understanding Images," *Proc. Image Understanding Workshop*, L. Bauman (ed.), Scientific Applications International Corp., McLean, VA, 1985.

Poggio T. and staff. "MIT Progress in Understanding Images," *Proc. Image Understanding Workshop*, L. Bauman (ed.), Scientific Applications International Corp., McLean, VA, 1987.

Poggio, T., H.L. Voorhees and A.L. Yuille. "Regularizing Edge Detection," *Artificial Intelligence Laboratory Memo 776*, Massachusetts Institute of Technology, Cambridge, MA, 1984.

Poggio, T., V. Torre, and C. Koch. "Computational Vision and Regularization Theory," *Nature*, **317**, 314-319, 1985.

Richards, W., and D.D. Hoffman. "Codon Constraints on Closed 2D Shapes," *Computer Vision, Graphics, and Image Processing*, **32**, 265-281, 1985.

Reichardt W. and T. Poggio. "Visual Control of Orientation in the Fly: I. A Quantitative Anaylsis," *Quart. Rev. Biophysics*, **3**, 311-375, 1976.

Reichardt W. and T. Poggio. "Visual Control of Orientation in the Fly: II. Towards the Underlying Neural Interactions," *Quart. Rev. Biophysics*, **9**, 377-439, 1976.

Rock, I. **Orientation and Form**, Academic Press, New York, 1973.

Terzopoulos, D. "Integrating Visual Information From Multiple Sources," In: **From Pixels to Predicates**, A.P. Pentland (ed.), Ablex Publishing Corp., Norwood, NJ, 1986.

Tikhonov, A.N. and V.Y. Arsenin. **Solution of Ill-Posed Problems**, Winston and Wiley Publishers, Washington, DC, 1977.

Torre, V. and T. Poggio. "On Edge Detection," *Artificial Intelligence Laboratory Memo 768*, Massachusetts Institute of Technology, Cambridge, MA, 1984.

Ullman S. **The Interpretation of Visual Motion**, The MIT Press, Cambridge, MA, 1979.

Ullman, S. "Visual Routines," *Cognition*, **18**, 1984.

Verri, A. and T. Poggio. "Motion Field and Optical Flow: Qualitative Properties," *Artificial Intelligence Laboratory Memo 917*, Massachusetts Institute of Technology, Cambridge, MA, 1986.

Voorhees, H.L. and T. Poggio. "Detecting Textons and Texture Boundaries in Natural Images," *Proc Intl. Conf. on Computer Vision*, Computer Society of the IEEE, Washington, DC, 1987.

Waxman, A. "Image Flow Theory: A Framework for 3-D Inference from Time-Varying Imagery," In: **Advances in Computer Vision**, C. Brown (ed.), Lawrence Erlbaum Assocs, NJ, 1987.

Wyllie, J.C. "The Complexity of Parallel Computations," *Technical Report 79-387*, Dept. of Computer Science, Cornell University, Ithaca, NY, 1979

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY
and
CENTER FOR BIOLOGICAL INFORMATION PROCESSING
WHITAKER COLLEGE

A.I. Memo No. 970
C.B.I.P. Memo No. 027
October 1987

VISUAL INTEGRATION AND DETECTION OF DISCONTINUITIES: THE KEY ROLE OF INTENSITY EDGES

Ed Gamble and Tomaso Poggio

Abstract: Integration of several vision modules is likely to be one of the keys to the power and robustness of the human visual system. The problem of integrating early vision cues is also emerging as a central problem in current computer vision research. In this paper we suggest that integration is best performed at the location of discontinuities in early processes, such as discontinuities in image brightness, depth, motion, texture and color. Coupled Markov Random Field models, based on Bayes estimation techniques, can be used to combine vision modalities with their discontinuities. These models generate algorithms that map naturally onto parallel fine-grained architectures such as the Connection Machine. We derive a scheme to integrate intensity edges with stereo depth and motion field information and show results on synthetic and natural images. The use of intensity edges to integrate other visual cues and to help discover discontinuities emerges as a general and powerful principle.

Acknowledgments. This report describes research done within the Artificial Intelligence Laboratory. Support for the A.I. Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research (ONR) contract N00014-85-K-0124. Support for this research is also provided by a grant from ONR, Engineering Psychology Division and by a Hughes Aircraft Corporation gift to the Artificial Intelligence Center for T. Poggio.

1 Introduction

One of the keys to the reliability, flexibility and robustness of biological visual systems is their ability to integrate several different visual cues. Early vision processes such as stereo, motion, texture, shading and color give separate cues to the distance of three-dimensional surfaces from the viewer and to their material properties. Integration of the evidence provided separately by these cues can provide a more reliable map of the surfaces and their properties than any single cue alone.

Thus visual integration is likely to be a key to understanding biological visual systems and to developing robust vision machines. Existing methods do not seem capable of providing a general solution. Standard regularization[2] provides a common framework for many early vision problems and leads to the minimization of quadratic energy functionals. If standard regularization is used to integrate information from different processes, the energy functional consists of the sum of quadratic parts, each associated with a separate process. This implies that the result is a linear combination of the different cues (possibly with space-varying coefficients). Linear combination - say of depth from stereo and from shading - does not seem, however, a flexible enough integration method. Even more important, no instances of standard regularization can handle discontinuities, because the solution space is restricted to generalized splines[21,2]. As we will explain later, we believe that detecting and representing discontinuities (for instance depth discontinuities) is a key part of the integration step[21].

To overcome these difficulties we have developed an extension of regularization that promises to deal simultaneously with discontinuities and with the integration of vision modules. This extension is based on the use of coupled Markov Random Fields[1], introduced recently by Geman and Geman[9] and extended by Marroquin, Mitter and Poggio[19]. The standard regularization method for vision is a special case of this new approach.

1.1 The Role of Discontinuities

One of the most important constraints for recovering surface properties is that the physical processes underlying image formation are typically smooth:

[1] A different, interesting approach has be explored by Blake[3]

depth and orientation of surfaces are *mostly* continuous and so are reflectance and illumination. The smoothness property is captured well by standard regularization. Surfaces and their properties, however, are not always smooth: they are smooth *almost* everywhere, but not at discontinuities. Lines of discontinuity are themselves usually continuous, relatively smooth, nonintersecting curves. It is critical to detect the discontinuities reliably, because they usually represent the most important locations in a scene: depth discontinuities, for instance, often correspond to the boundaries of an object or of a part. Furthermore, discontinuities play a critical role in fusing information from different physical processes. The reason is clear: in smooth regions, the physical processes are coupled together by the imaging equation, and all contribute to image formation. However, the coupling is difficult to know precisely: it depends on quantities such as the form of the reflectance function. The effects of discontinuities are instead robust and qualitative: for instance, depth discontinuities usually correspond to intensity edges. Therefore, discontinuities are ideal places for integrating information. Furthermore, *partial* information about discontinuities in a single process can be detected relatively easily. Several types of motion discontinuities, for example, can be measured with simple operations on the time-dependent intensity array, especially if the interframe interval is small. Partial albedo discontinuities also are often detectable using simple operations. Intensity edges are detected quite reliably by the Canny edge detector. However, the fast, rough detection of discontinuities performed by these early operations is noisy and incomplete: it must be refined by integrating them across processes and by exploiting constraints on the continuity of discontinuities.

In summary, discontinuities: 1) represent the most useful information, 2) are easy to detect (though in a partial and possibly noisy way) and 3) provide good locations to integrate different cues.

1.2 Coupled Markov Random Fields

Markov Random Fields for image modeling have seen increasing use since the work of Geman and Geman[9]. Their utility for image modeling derives from several MRF characteristics. MRFs provide a natural way to impose general image properties of smoothness and continuity, for example of depth and motion, while also incorporating discontinuities. Bayes' rule establishes a relationship between the possibly corrupted observed data and the desired scene data. Solution methods are available, though often time consuming. Some recent MRF applications have involved scene segmentation using depths[18], texture[6] and motion[20].

A Markov Random Field on a lattice can be represented as a lattice of sites, each one with a random variable. The value depends probabilistically on the value of neighboring sites. The rules governing this local dependence can be given in a variety of ways and can be made to capture constraints such as the continuity of a surface (if the MRF represents depth values).

Our idea is to associate a MRF on a lattice to each physical process to be integrated and another (binary) MRF to its discontinuities (see figure 1). The lattices are coupled to each other to reflect the interdependence of the corresponding processes in image formation. Thus the various MRFs mirror the different physical events that underlie image formation: surface and surface discontinuities, spectral albedo and albedo discontinuities, shadows, surface normal, and so on. Physical constraints apply to each of these processes independently. In addition, there are constraints between these processes (for instance between depth and surface normal). The image data constrain the way the processes combine. Note that consideration of sequences of images in time will introduce additional powerful constraints such as rigidity. The constraints on the surfaces are local conditions (such as smoothness, necessary mainly because of its regularizing role in the face of omnipresent noise) valid everywhere *except* at discontinuities. As we discussed earlier, discontinuities are critically important and should be detected early.

Notice that the coupling of the line process with the associated continuous process provides a module that *combines region-based with boundary-based segmentation* (see figure 1).

The local potentials underlying the *a priori* probability distribution of the MRFs represent the constraints on the physical processes (smoothness, positivity, values within certain bounds, etc.); the coupling between MRFs represents the compatibility constraints *between* processes. The device of coupled MRFs provides an ideal tool to impose local constraints such as smoothness, allowing at the same time an explicit role for discontinuities through the *line processes*[9] and similar processes such as *occlusions*[19]. Our new idea is to incorporate additional *observable* discontinuity data provided by algorithms specialized to detect sharp changes in the observed properties of intensity, motion, stereo disparity, texture, and so on. The observable discontinuities

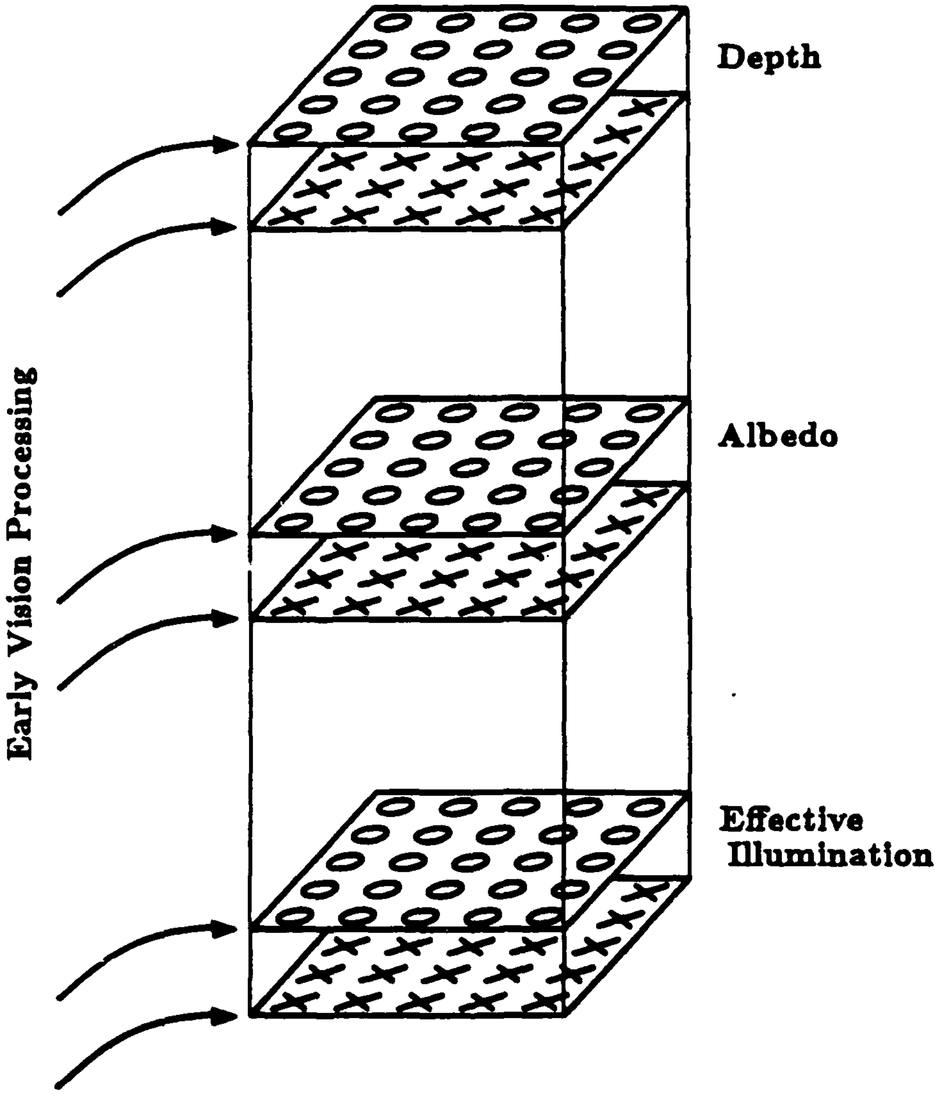


Figure 1: MRF lattices representing the output of different early processes and their discontinuities (the crosses represent the sites of the binary line processes). Each representation, for instance depth, is coupled to its discontinuities and to other cues such as intensity or motion.

provide an initial rough solution to the segmentation problem. Using the MRFs for estimating the fields gives increasingly precise solutions, simultaneously *filling in* the continuous regions that are only sparsely observable. The solution at each iteration is available to later modules, such as recognition.

1.3 The Key Role of Intensity Edges

One of the results of our integration work is that intensity edges play primary role in guiding the search for discontinuities in other processes (for instance depth). The point seems so important that we would like to phrase it as a rather general conjecture on the proper organization of the integration stage: ***intensity edges guide the detection of discontinuities in the other physical processes, thereby coupling surface depth, surface orientation, shadows, specularities and surface markings to the image data and to each other.***

The reason for the critical role of intensity edges is intuitively clear - usually changes in surface properties (depth, orientation, material, texture) produce large intensity gradients in the image. Under the assumption of opacity and of a simple imaging model (the reflectance function is assumed to contain a lambertian and a specular term), there are six physical causes for large intensity gradients in the image: occluding edges (*extremal edges* and *blades*), folds, shadow edges, surface markings and specular edges. In addition, motion discontinuities are usually coupled to intensity edges. It is for exactly this reason that edge detection is so important in artificial - and probably also biological - vision.

1.4 Plan of the Paper

In this paper we introduce a method for detecting and reconstructing depth discontinuities by using the information provided by intensity edges. We do the same for motion discontinuities. First we introduce the Markov Random Field formalism. The use of intensity edges for surface interpolation is discussed next, together with the derivation of the associated MRF model. We then describe our Connection Machine implementation and the results on synthetic and real data. Finally the discussion focuses on the open problems and on the implications of our results for the general problem of integrating all vision modules.

2 Coupling Intensity Edges with Sparse Depth Data

To illustrate our approach we consider the specific and important problem of computing an approximate surface and especially the surface depth discontinuities from sparse depth data[10,25,18]. The main new idea here is to exploit the integration of additional vision cues. In particular we describe a scheme in which intensity edges are integrated with sparse depth data. Sparse depth data arise from the output of feature–based stereo algorithms. Typical stereo algorithms provide depth data at a subset of image features[15,10,8]. These features might be a Laplacian filter's zero-crossings from one of the intensity images. The depth information is computed by measuring pixel displacements (disparity) between corresponding image features. As is typical of all known stereo algorithms, the disparities are plagued by errors precisely at depth discontinuities where surfaces are usually occluded.

The problem, then, is to smooth and fill in the sparse depth data (i.e., reconstruct the surface), while detecting the critically important depth discontinuities. Prior attempts at depth discontinuity identification allowed the discontinuities to form anywhere in the image provided the depth difference between neighboring sites was significant[18,24]. Due to the sparseness and noise in the depth data, the identified discontinuities are: 1) offset from and 2) ragged or wiggly compared with the *correct* discontinuities. These limitations become more serious when the images contain a large range of depth differences, as in natural images.

Because of the constraints on image formation discussed earlier, the *correct* depth discontinuities will, in almost all cases, correspond precisely to the locations of intensity edges. Our integration scheme exploits this by restricting depth discontinuity formation to a subset of the intensity edges. This restriction ensures that the smoothness and continuity of discontinuities can be no worse than the intensity edges themselves. In addition, the difficult problem of MRF parameter specification is simplified since this integration scheme proves less sensitive to MRF parameter variations, particularly when the depth data contain a large range of depth differences.

There are some cases in which discontinuities will not occur at intensity edges. Any object that *blends in* with its background presents such a case. This situation occurs rarely in natural scenes; yet, for practical reasons such as camera underexposure or saturation, the object may blend in with the background at some locations. However, for these cases, the point is somewhat moot, since without intensity edges, feature-based stereo or motion algorithms will not provide depth or motion data.

A more general situation arises when the features used for stereo or motion are different from the discontinuity–limiting features. This is desirable since the continuity constraints used by stereo and motion algorithms assume that the features used for matching are located on surfaces. Thus stereo and motion algorithms should use high resolution, dense features that identify surface markings as opposed to bounding contours which in general correspond to surface locations that are different in the two images of a stereo pair. The discontinuity–limiting features however can be chosen to better correspond to object boundaries.

The results section contains examples in which the discontinuities are identified and the surface reconstructed both with and without the benefit of intensity edge information. The next section presents a limited overview of MRF particulars and contains the appropriate MRF energy function for integrating intensity edges with, in this case, the sparse depth data produced by a stereo algorithm.

3 MRF Formulation for Stereo and Intensity Edge Coupling

The theory of Markov Random Fields can be found elsewhere[9,17]. We present only an overview here followed by a description of the energy functions used for integration.

The Hammersley-Clifford theorem states the equivalence between a MRF and a Gibbs distribution as follows. If X is a MRF on a lattice S with respect to the neighborhood system G, then $P(X = \omega)$ is given by:

$$P(X = \omega) = \frac{1}{Z} e^{-\frac{1}{T} U(X)} \qquad (1)$$

Z is a normalization factor, T is the *temperature* and $U(X)$ is the *energy function*. The temperature parameter, T, could be absorbed into $U(X)$; however, when the solution method is discussed, T proves useful as a separate variable. The energy function is of the form:

$$U(X) = \sum_C U_C(X). \tag{2}$$

The sum of the *potentials*, $U_C(X)$, is over the neighborhood's *cliques*. A clique is either a single lattice site or a set of lattice sites such that any two sites belonging to it are neighbors of one another. The function $P(X = \omega)$ is called the *prior* distribution and abbreviated here by $P(X)$.

The prior distribution on X, where X, for example, might be the reconstructed surface, must be determined based on some observations or input data, Y. To relate X to Y Bayes' formula is used,

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}. \tag{3}$$

The observations, Y, are obtained conceptually by degrading X, such as by the addition of noise or blurring. If the type of degradation is known, the distribution $P(Y|X)$, can be computed. Marroquin[17] has shown that for the case of zero-mean white Gaussian noise, $P(Y|X)$ is a Gibbs distribution with potential:

$$U(Y|X) = \sum_{i \in S} U_i(Y|X); \quad U_i(Y|X) = -\alpha\gamma_i(x_i - y_i)^2. \tag{4}$$

The sum is over all lattice sites and

$$\gamma_i = \begin{cases} 1, & \text{if input data exists at lattice site } i \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

When this result for $P(Y|X)$ is combined with the MRF prior distribution, $P(X)$, and Bayes' rule the *a posteriori* distribution $P(X|Y)$ is:

$$P(X|Y) = \frac{1}{Z} \exp\left\{-\frac{1}{T}\sum_i U_i(X|Y)\right\} \tag{6}$$

for $U_i(X|Y) = U_i(X) + U_i(Y|X)$ and with Z a normalization constant independent of X. This *a posteriori* distribution provides the likelihoods for all possible states X, given the observable data Y.

Given the posterior distribution $P(X|Y)$ and the *external field* Y the desired field X can be retrieved once a suitable error criterion is specified. The

Maximizer of the Posterior Mean (MPM) reduces the problem of annealing and has been successfully applied for our results. With the criterion specified, the relaxation algorithm for solution is largely determined. The question of a suitable error criterion and algoritnmic consequences has been thoroughly discussed by Marroquin[17].

The problem has now become one of specifying the MRF potentials, $U_i(X)$ and $U_i(Y|X)$. The potentials impose the physical constraints of continuity and smoothness of surfaces (except at depth discontinuities) along with continuity and smoothness of depth discontinuities. These constraints are imposed by tailoring the energy function to minimize the energy (maximize the probability) when the state occupied satisfies the desired physical constraints. Typically this choice is empirical although one might envisage estimating the *prior* associated with, for instance, depth smoothness from a specific class of surface data.

The MRF state space used herein is similar to that of Geman and Geman[9] along with Marroquin[17] where each lattice site is composed of a depth process and two line processes, $X = \{F, L\}$. The depth process, F, is a continuous random variable whose value is related to the distance of a surface point from the observer. The value of F at site i is denoted as f_i where $-\infty \leq f_i \leq \infty$. The depth process neighborhood system to site i consists of the four nearest neighbors: east, south, west and north, to i. Although a continuous random variable should not be updated using the Heat Bath algorithm, the depth process can be deterministically updated[17], provided the MRF energy is suitably defined. Figure 2 illustrates the MRF lattice with the depth and line processes.

The line process used here, L, contains a vertical and horizontal orientation that are conceptually located between lattice sites. The vertical line process is located between its lattice site and the neighboring eastern lattice site, whereas the horizontal line process separates its lattice site and the nearest southern lattice site. Each orientation is a binary random field, $l_i^j \in \{0, 1\}$ where the scripts on l_i^j denote the line process that separates lattice site i from j. The horizontal line process at site i is denoted as l_i^h; the vertical line process is l_i^v. Smoothing of the depth process is inhibited when the line state is *on*, $l_i^j = 1$, since smoothing should not occur across depth discontinuities; otherwise, depth process smoothing is performed. An *on* state signifies the presence of a depth discontinuity. The conditions for

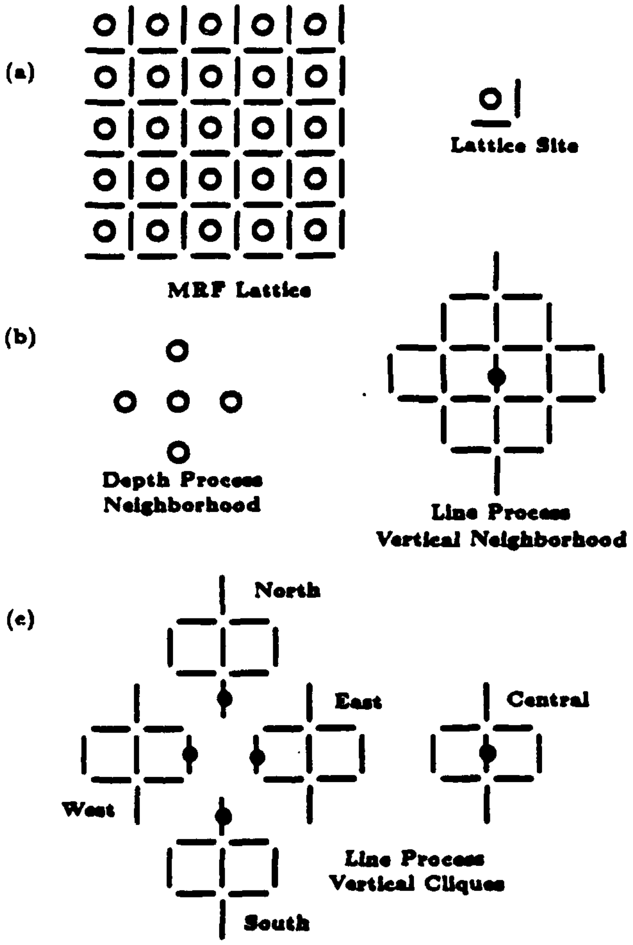


Figure 2: (a) A lattice site is composed of a single depth process (illustrated with a circle) along with a vertical and a horizontal line process. The MRF Lattice consists of a rectangular grid of these lattice sites. (b) The neighborhood for the depth process and the vertical line process neighborhood. The black dot in the line process neighborhood indicates the lattice site for this neighborhood. (c) The five maximal cliques (north, east, south, west and central) for the vertical line process are shown. In this paper we only consider configurations of the central clique. This is equivalent to assigning zero energies to all configurations of the other four cliques.

depth discontinuity formation are encapsulated in the MRF energy function presented subsequently.

The external fields to the MRF are the sparse depth information and the intensity edges. The sparse depths, G, are represented by two variables, g_i and γ_i for site i. The value g_i is analogous to f_i; it is continuously valued over the real numbers, although in practice, since g_i is provided by stereo output, it is discrete. The variable γ_i encodes the sparseness of the stereo output and is defined as in equation 5.

The intensity edges are represented by the field, E. This field is similar to the line process, L, except that $e_i^j = 1$, rather than indicating the presence of a depth discontinuity, *permits the formation* of a depth discontinuity between lattice site i and neighbor j. The MRF energy is designed so that $e_i^j = 0$ implies (in the present implementation) $l_i^j = 0$ for all $i, j \in S$. An edge detector, such as Canny's[4], will mark *a site i* as an edge, but e_i^j marks potential discontinuities *between sites i* and j. To resolve this ambiguity, if an edge is at site i, then $e_i^k = 1$ where k is each of the nearest neighbors to site i. This intensity edge field, E, along with G comprise the MRF external field Y such that $Y = \{G, E\}$.

Given the external fields, Y, and the random variables, X, equation 6 provides the posterior distribution with the MRF energy given as

$$U(x|y) = \sum_i U_i(x|y)$$

$$U_i(x|y) = \alpha\gamma_i(f_i - g_i)^2 + \sum_{j \in nn} (1 - l_i^j)(f_i - f_j)^2 + \sum_{j \in <h,v>} \left[\beta U_C(l_i^j) + \beta'(1 - e_i^j) l_i^j\right]. \tag{7}$$

The first term in this equation is the coupling between the depth process and the sparse and noisy input data. The coupling factor, α, is related to the noise in g. For noiseless data, $\alpha \to \infty$ thereby ensuring $f_i = g_i$. Otherwise, when $\alpha = 0$ no input data coupling occurs and f is smoothed by the term involving $(f_i - f_j)^2$ in equation 7. The precise relation between α and the noise depends on the noise model assumed. For a model of measurement that includes Gaussian random noise

$$\alpha = \frac{1}{\sigma^2}$$

where σ is the gaussian's half width at half maximum[17]. Note that if the noise model's parameters vary locally, it might be appropriate to vary α locally as

$$\alpha_i = \frac{1}{\sigma_i^2}.$$

Local variation in noise parameters does occur in the stereo algorithm of Drumheller and Poggio[7]; this variation is reflected in the stereo match scores of that algorithm. The present paper does not address this issue; here we keep α constant, usually in the range 0.1 to 2.0. The input data coupling to f occurs when $\gamma = 1$. Typically 5 to 10% of the lattice sites have input depths associated with them.

The last term in equation 7 implements the integration scheme between sparse stereo depths and intensity edges. The term forbids depth discontinuity formation except where an external edge exists. Discontinuity formation is prevented by letting $\beta' \to \infty$. When $l_i^j = 1$ and $e_i^j = 0$, this term contributes a large energy, $U_i(x|y) \to \infty$ and the associated probability for $l_i^j = 1$ is zero. At sites where $e_i^j = 1$ this energy term contributes nothing and the depth discontinuity formation is determined by the other factors in equation 7. The problems of misalignment might be handled by suitably modifying this term in the energy $U_i(x|y)$ to produce a it cone of influence or, for a simple case, by "thickening" the input intensity edges. For instance, we may use instead of e_i^j in equation 8, $e_i^j * G$, where $*$ denotes convolution and G is a gaussian or another appropriate cone of influence function. The results presented in this paper do not utilize a cone of influence.

The second and third terms in equation (7) encapsulate our prior expectations concerning depth discontinuities and surface reconstruction. They compose the potential $U(X)$ of the prior distribution (equation 1). These two terms 'compete' in the sense that turning on a line costs energy $\beta U_C(l_i^j)$ but saves energy $(f_i - f_j)^2$. The interplay of these two potentials largely determines the formation of depth discontinuities where $e_i^j = 1$. The second term couples the line and depth processes, the third term determines the line process clique energy. This line and depth process coupling is summed over the nearest neighbors, nn, to site i, with each neighbor contributing an energy $(f_i - f_j)^2$ when $l_i^j = 0$.

The quadratic term, $(f_i - f_j)^2$, tends to smooth the depth process since it is minimized when $f_i = f_j$. Depth discontinuities have a higher probability of forming when the energy to create a line, $\beta U_C(l_i^j)$, is less than this energy to smooth the depths. The factor β is a free parameter that determines what size depth difference is likely to produce a depth discontinuity. Specification of β is largely image dependent and, although a suitable range has been determined, a general theory specifying β remains elusive. The line process clique energy will be examined in detail later.

The Heat Bath algorithm cannot be simply applied to equation 7 since the f_i are continuous variables. Instead we employ a technique to smooth the depth process deterministically, but to update the line process stochastically with the Heat Bath algorithm[17]. With the line process state fixed, the MRF energy of equation 7 is non-negative definite quadratic with a stable and unique fixed point for the f_i (practically, β' never contributes since the configuration $e_i^j = 0$ and $l_i^j = 1$ has a vanishing probability). In this situation, the depth process can be smoothed deteministically to find the fixed point. After this fixed point in depth is determined, the line process is stochastically updated, the new fixed point in depth is determined and the scheme is repeated.

Once the line process approaches equilibrium (roughly 1000 iterations), statistics are gathered to compute the MPM estimate. The MPM estimate is computed from $P(l_i^j = 1) = \frac{1}{n}\sum l_i^j$, where n is the number of iterations over which statistics are gathered[17]. When $P(l_i^j = 1) \geq (0.5 + 1/\sqrt{n})$, statistical fluctuations about 0.5 are reduced and the MPM estimate is turned on to mark a discontinuity. Use of the MPM estimate does not require annealing but the *a posteriori* distribution's coupling parameters must produce a reasonable amount of line process agitation thereby sampling much of the line process sample space.

3.1 Choice of Line Clique Energies

Figure 2 shows the line process neighborhood for the vertical line process. Of the five cliques shown for this neighborhood, only the clique centered about the vertical lattice site has, by design, a non-zero potential $U_C(l_i^j)$. This potential depends on the 256 possible configurations associated with the clique. The desirable configurations are a small subset of all possible configurations and they impose the constraints of smoothness and continuity on the depth discontinuities. These constraints are embodied in the following five heuristics which divide the desirable configurations into *classes*:

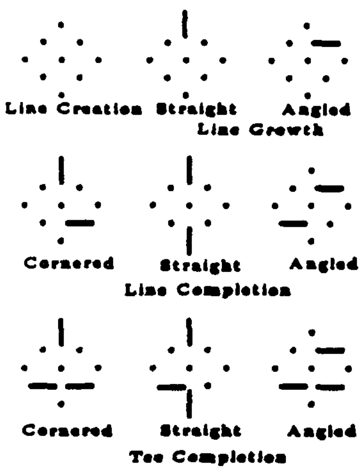


Figure 3: The four classes of non-forbidden line configurations for the vertical line process. A dot, '.' represents an off state; on states are shown with their oriented lines. The symmetry operations producing the other allowed configurations are discussed in the text. The horizontal line process configurations are identical provided the vertical line process cliques are rotated by 90 degrees.

- Turn on a lone site provided a 'large' depth discontinuity is present [*Line Creation*].
- Turn on a site extending an already present line segment even if the depth discontinuity is 'small' [*Line Growth*].
- Always turn on a site if doing so would connect two line segments [*Line Completion*].
- Allow tees to occur infrequently where supported by at least a 'small' depth discontinuity [*Tee Completion*].
- All other configurations should occur rarely if at all [*Forbidden*].

Examples of the first four classes are shown in figure 3. In addition to these configurations, three symmetry operations produce the other non-forbidden classes. These symmetry operations are: rotation by 180 degrees about an axis perpendicular to the page, reflection about the vertical axis (for the vertical line process orientation) and the 180 degree rotation followed by the reflection operation. With these symmetry operations and clique classes, a total of 22 unique configurations are allowed from the original set of 256. When $l_i^v = 0$ (line is off), the clique potential is 0. However, when $l_i^v = 1$, the clique energy is determined by the five classes; this is the energy required to turn on the line.

The line process clique considered here is only one of the cliques associated with the neighborhood shown in figure 2. In previous work[9,17], the smaller neighborhood did not readily produce lines of any orientation; the cliques tended to create vertical or horizontal line segments. The 'large' neighborhood used here (though incompletely, because we assign zero energies to several cliques), does encourage isotropic line formation without exacting too high a computational penalty.

4 Stereo and Synthetic Image Results

The MRF scheme for coupling intensity edges to sparse stereo depth data has been implemented on a Connection Machine[11]. The sparse depth data and intensity images from both real stereo and synthetic images have been examined. This section presents these image results for some typical images.

4.1 Connection Machine Implementation

The Connection Machine (CM) is a fine-grained parallel computer manufactured by Thinking Machines Corporation. We used their CM-1 model with 16k processors. Each processor is connected to its four nearest neighbors (north, east, south and west) in a two-dimensional grid, the NEWS network, and each 16 processor group is connected to a 12-dimensional hypercube, the Router. These two communication modes allow fast access between neighboring processors and logarithmic-time access between any two processors. Each processor is a simple 1-bit processor with 4 kilobits of memory. All processors execute a single instruction stream. The CM was configured to match the image size, 256 x 256, by using virtual processors.

For the MRF implementation each CM processor represents an MRF lattice site. This configuration proves ideal for implementing the MRF cliques over the CM NEWS network. The limited number of non-forbidden line clique states and energies are stored in tabular form at each processor. Determination of the line clique state requires access to the four nearest neighbors plus the north-east (south-west) neighbor for the vertical (horizontal) orientation. At the image borders, the line processes are always *on*, thereby conveniently preventing depth process smoothing beyond the borders.

The MRF input data was obtained from two previously implemented CM-1 algorithms. For the real stereo depth data, MIT's Eye-Head system provided the stereo pair and the Drumheller-Poggio CM-1 stereo algorithm[8] produced the disparity data at a subset of DOG zero-crossing features. The intensity edges came from Todd Cass' [13] implementation of Canny's edge detector. These edges do not coincide with the stereo algorithm features.

When synthetic data was used, the image depths were produced by the TMC 3-D Toolkit as was a dense depth map. A sparse map was obtained by randomly discarding 90 to 95 percent of the depth values. Uniformly distributed random noise was added to the synthetic sparse depth data.

The initial line process state is set to mimic the intensity edge map as provided by the Canny edge detection stage. The MRF depth values are created by using the sparse input depths to "brush fire fill" and then by deterministically smoothing the depth values. During the deterministic smoothing of the initial depth process, the depth external field coupling, α, is infinite.

4.2 Results

Figure 4 shows the MRF results on a synthetic image for two intensity edge coupling schemes. In the first scheme, intensity edges are not used in the MRF process. This allows depth discontinuities to form anywhere and is achieved by setting $e_i^j = 1$ for all $i, j \in S$. The upper left image shows the synthetic scene from which the sparse depth data was derived. The lower left image in Figure 4 illustrates the depth discontinuities identified with the MPM estimate of the MRF process. When the depths vary rapidly, many closely spaced discontinuities are formed. These discontinuities are ragged and also displaced from the actual object boundaries (as marked by intensity edges). The reconstructed depth surface is not shown.

The second scheme strongly penalizes depth discontinuity formation ev-

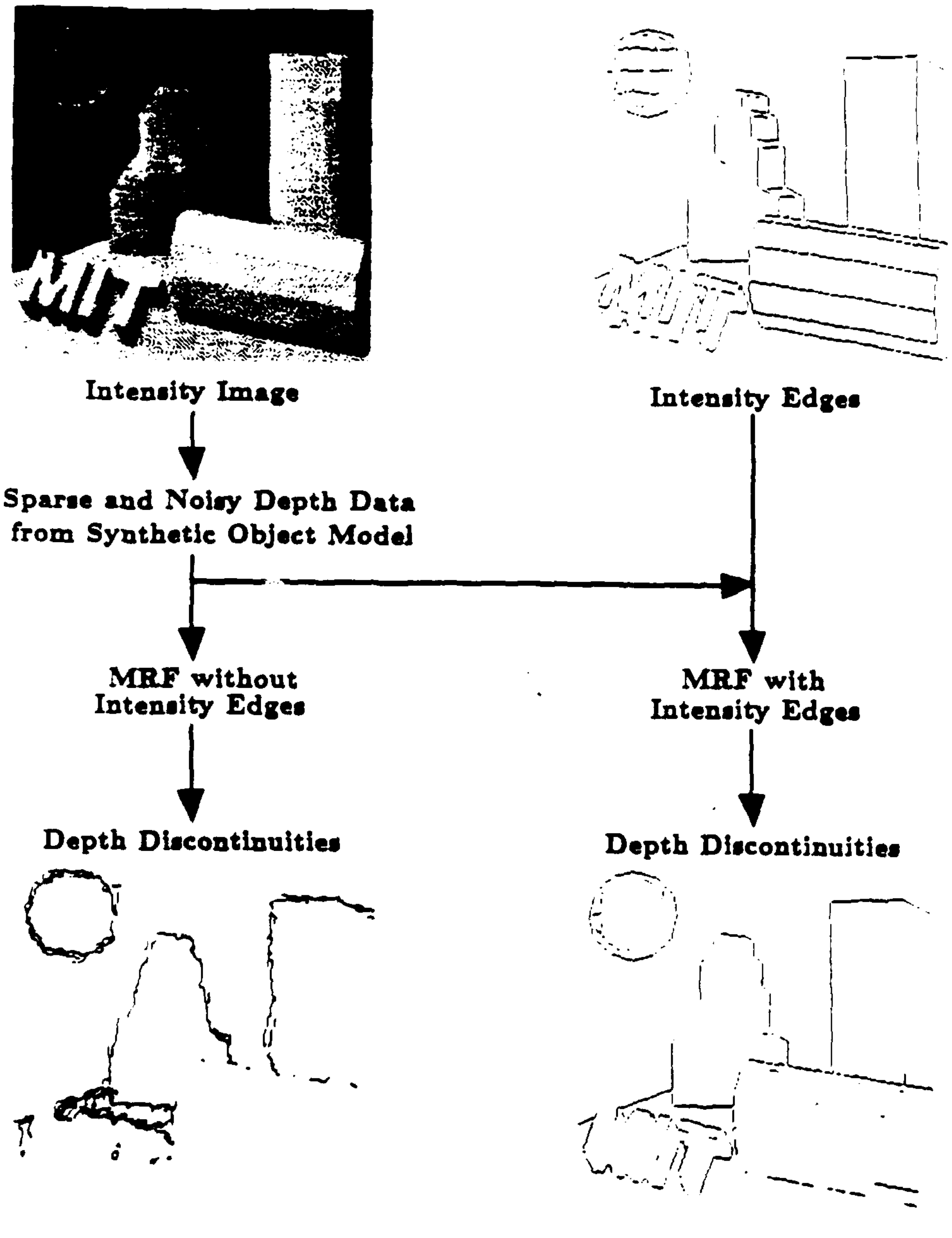


Figure 4: The MRF process and its result on a synthetic image. Almost all depth discontinuities are found when intensity edge coupling is utilized. The steepness of the geodesic dome's boundary leads to false discontinuity identification.

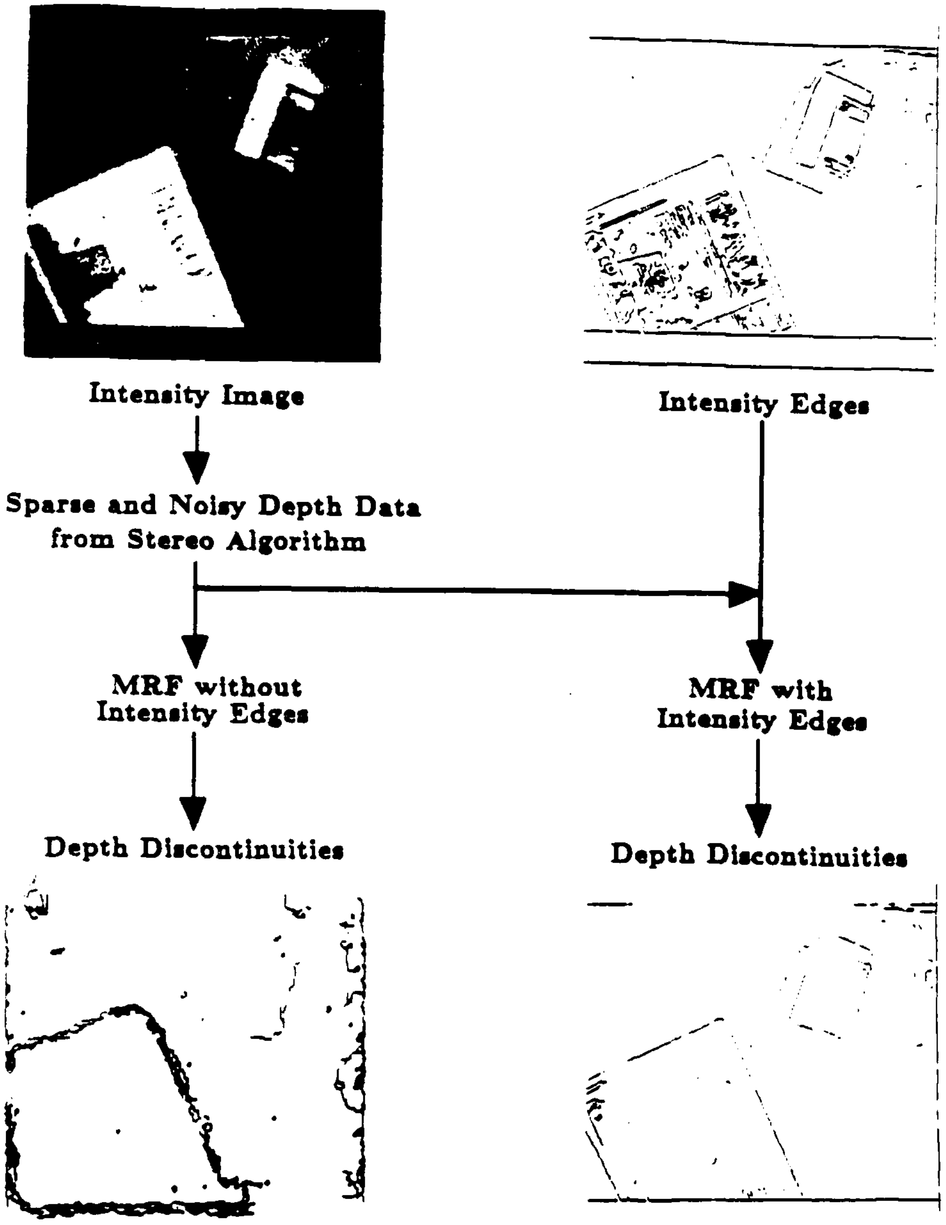


Figure 5: The MRF process and its result on a real image with computed stereo data. For both cases the texture on the newspaper has disappeared; however, without intensity edges, the small box on the upper right also disappears. When intensity edges are used some of the box's borders persist and the newspaper border is well localized.

erywhere except at the intensity edges shown in the upper right image of Figure 4. The external field, e_i^j, equals one only at the intensity edges pixels. The depth discontinuities found are shown on the lower right of Figure 4. Nearly all the intensity edges due to surface orientation and texture are eliminated. In some places, such as near the geodesic sphere's boundary, the surface slope alone is large enough to yield a depth discontinuity.

Another representative image–this time a real image–is shown in Figure 5 where a stereo algorithm produced the sparse depth data. The right image from the stereo pair appears on the upper left of Figure 5. This scene consists of a tall stack of newspapers and a small box or carton. The stereo depth data and the reconstructed surface are not shown. Once again we consider two cases, depending on whether or not the intensity edges are utilized. Without the intensity edges, as with the synthetic stereo results, the depth discontinuities are poorly positioned and ragged. However, with the intensity edges (upper right of Figure 5), the discontinuities on the lower right agree reasonably well with the object boundaries.

For these stereo image results, a few difficulties are worth mentioning. A large depth discontinuity along the top left of the newspaper boundary is not found. The stereo algorithm produced very poor depth data at this location and positioned the depth change roughly 5 pixels above the newspaper intensity edge used by the MRF process. Also the small box's shadow yielded a small disparity that created a depth discontinuity. The box itself also had a small disparity so that modifying MRF parameters to eliminate the shadow discontinuity would have eliminated the box's discontinuity. This sort of variability is inevitable until a reasonable method for local parameter estimation is developed.

Situations can arise wherein discontinuity detection is hampered when the intensity edge sites do not coincide with the sites at which external depth data are provided. Figure 6 displays a possibility where a depth discontinuity should form between features A-1 and A-2 inclusive. However, the discontinuity can only form on the intensity edge at B-1 and, because of depth filling and smoothing, the discontinuity may be *washed out*. The *washing out* depends primarily on the depth difference, the separation between edges A-1 and A-2 and the smoothing parameters. If edge B-1 were on A-1 or A-2, then the discontinuity could form readily. One approach to avoid this coincidence problem is to project a cone of influence about the intensity edge

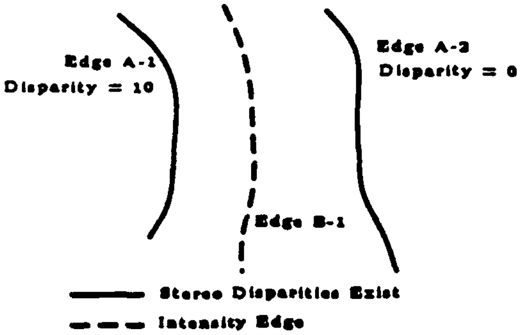


Figure 6: The disparities at edges A-1 and A-2 suggest that a depth discontinuity should be formed somewhere between A-1 and A-2. Yet, because of depth process smoothing, the depth difference at intensity edge B-1 may be too small to support a discontinuity. No discontinuity will form due to this 'misalignment' of edges.

location. Then the discontinuities could form not only at the intensity edges but also for one, two or more pixels on either side of the edge. This has the disadvantage of leading to somewhat poorly localized and ragged edges. Straightness of the resulting line process is enforced locally by the intrinsic prior of the line process when the cone of influence is no larger than the line process neighborhood. Another approach, used here, was to avoid the *washing out* by an appropriate selection of the coupling parameters. More work must be done in this area.

5 Coupling Intensity Edges to Sparse Motion Data

The simplicity of limiting discontinuities to a subset of intensity edges immediately suggests its use for other vision modules. The same principles employed for the stereo depth application have been utilized on motion data. As with depths, motion fields both from synthetic data and a feature-based motion algorithm have been used to identify motion discontinuities and to smooth and fill the sparse motion field. The difference is that motion is a vector field; depth is not.

The MRF energy of equation 7 is modified by replacing the random field variable, F, by a vector random field, $\vec{M}$. Likewise, the external field, G, becomes a vector field, $\vec{N}$. The MRF energy is:

$$U_i(x|y) = \alpha\gamma_i|\vec{M}_i - \vec{N}_i|^2 + \sum_{j\in nn}(1 - l_i^j)|\vec{M}_i - \vec{M}_j|^2 + \\ \sum_{j\in<h,v>}\left[\beta U_C(l_i^j) + \beta'(1 - e_i^j)l_i^j\right] \quad (8)$$

where $\vec{M} = u\hat{e}_x + v\hat{e}_y$ with a similar definition for $\vec{N}$ and where $|\vec{M}_i - \vec{M}_j|^2 = (u_i - u_j)^2 + (v_i - v_j)^2$. The input field $\vec{N}$ contains the two components of the optical flow; the output is $\vec{M}$ or equivalently, (u_i, v_i) for all lattice sites i. With this energy formulation, motion field direction discontinuities are not identified, only magnitude discontinuities are marked.

A specialized motion algorithm, such as Horn and Schunk's[12], can be used to compute the motion field for input to the MRF. The motion data employed here derive from a parallel algorithm[14] that provides match scores much like the previously used stereo algorithm. Match scores provide a local measure of trust for the motion data but are not utilized here. Rather than splitting the problem into early and middle vision parcels, an alternative approach uses the MRF machinery to compute the motion field in addition to segmenting the images[20].

Figure 7 illustrates some results on a simple synthetic motion sequence. The image contains a white square with a small grey texture marking moving diagonally across a grey and black background. The motion field is non-zero only on the white square and its texture marking where both x and y components exist. Roughly 5% of the image motion data is input to the MRF. The bottom half of figure 7 shows the motion discontinuities identified both with and without intensity edge information. Again, the intensity edges significantly enhance the localization of "nice" motion discontinuities.

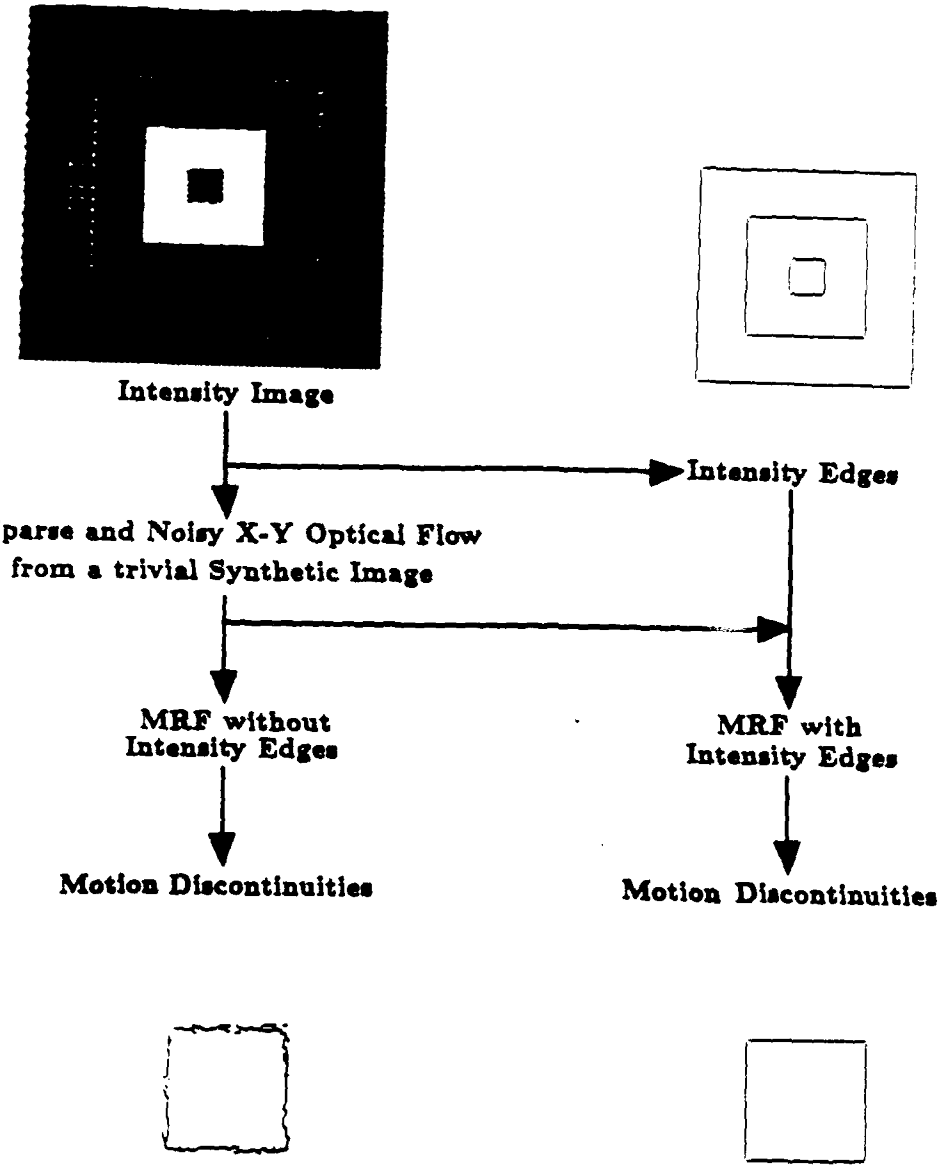


Figure 7: The MRF process and its result on synthetic motion data. Motion data exists at only 5 percent of the image pixels.

6 Discussion

6.1 Central role of intensity edges

The results presented here support the idea that intensity edges can be used as the primary cue to help detect, complete and precisely locate the discontinuities in the other processes such as depth, motion, texture and color. As we mentioned earlier, the reason for this is that discontinuities in depth, surface orientation, motion, texture and color typically originate large gradients in the image intensity, i.e. edges. Texture boundaries, for instance, can be synthesized without any intensity edge; it is sufficient to look around to convince ourselves that in the real world most of the texture boundaries occur together with an intensity edge. The same is true for motion discontinuities. Color boundaries also correspond to brightness boundaries (isoluminant borders exist only in the psychophysics lab!). In addition intensity edges can be better localized than motion, depth, texture and color discontinuities. The case of texture is especially clear: the uncertainty in the location of texture boundaries is no less than the size of the basic elements of texture, called textons[26] and usually several times as much. In most cases stereo cannot provide precise depth discontinuities because of occlusions. Color is in a similar situation because of the coarse scale at which it is computed (the low resolution is imposed by the low signal to noise ratio and the desired insensitivity to small surface markings).

Psychophysics also suggests that intensity information has a privileged role relative to other cues. Cavanagh[5] has shown that only intensity edges can support subjective contours and shadow interpretation. Furthermore, discontinuities portrayed through cues besides intensity edges, are more difficult to see at the level of recognition.

6.2 Open problems in the approach

The preliminary results obtained by integrating intensity edges with depth and motion data are encouraging, as the figures show. There are, however, many open questions that have to be answered before our theory can be regarded as a serious first step towards understanding visual integration. First, there is the question of the overall organization of the integration stage, the nature of the interactions and the couplings between the different cues. There are also more specific questions about our technique of visual integration and discontinuity detection.

6.2.1 The Structure of Visual Integration

The scheme sketched in figure 8 is a preliminary suggestion for the structure of visual integration. It is close in spirit to the ideas about intrinsic images proposed by Barrow and Tennenbaum[1]. They did not, however, have the powerful theory of coupled MRF models to implement their ideas.

Information about the image intensity has a primary role – intensity edges help the line processes associated with color, texture, motion and depth. Depth itself has also a special role – in a sense, it is the main output of the whole system. Motion, texture and color are coupled to depth. They may not be directly coupled to each other. Notice that *the main couplings are through the line processes*, according to the principles outlined in the introduction. Notice also that local estimates of reliability may be used to control locally the strength of the coupling: we have seen earlier that in the MRF model the coupling between depth and its discontinuities is controlled by the parameter α which is inversely proportional to σ^2.

The line processes may receive data from early algorithms – at this point it is an open question how. In the present implementation the intensity edges are totally driven by external data provided by the Canny edge detector whereas depth and motion do not get external information about discontinuities in depth or motion.

The intensity edges are also coupled with a higher level field that favors configurations of the subjective contour type, providing completion of lines and collinearity on a more global basis than the neighborhood of the line process[22]. The depth line process is coupled with another high-level field that provides the correct constraints on the interactions between contours of overlapping objects. A **T** junction is a clue to occlusion by one of the two surfaces bounded by it; an **X** intersection indicates that one of the surfaces may be transparent. The high-level features couple these configurations of the line process to the appropriate states of the depth process. If no values are locally available, default values for *in front* and *behind* are given to the depth process.

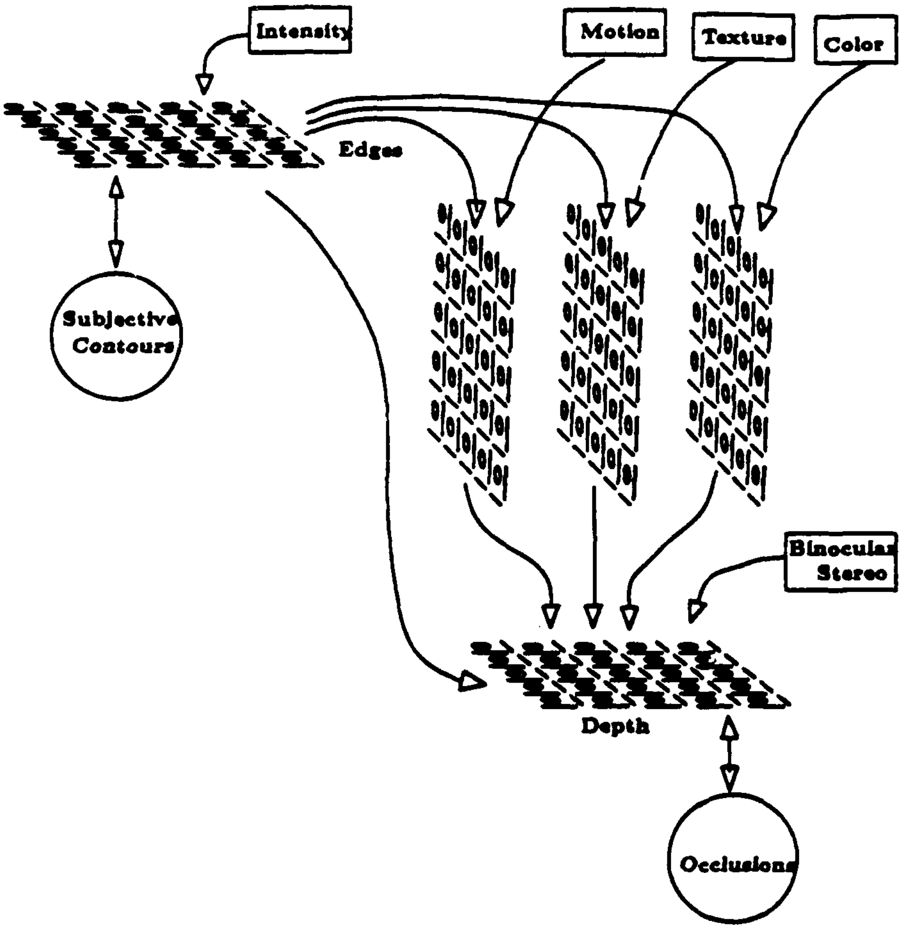


Figure 8: The organization of the integration stage. Each of the processes is coupled to its line process. Intensity data feed into the motion, color, texture and depth line processes. The line processes are not hidden processes: they may also receive data from specialized discontinuity detectors. The intensity line process gets input data from Canny edges. It is coupled to a higher level field which implements constraints of line continuation and collinearity on a more global basis than the neighborhood system of the line process. The line process associated with the depth process is also coupled to a higher level field which implements the appropriate constraints underlying occlusions of surfaces. The plausibility of interactions between motion, texture and color is an open question.

6.2.2 Detailed Questions

Other open questions are: *integration of additional visual cues, local vs. global constraints on the line process, tolerance in registration, multiresolution fields, approximative algorithms and neural implementations* and *learning of parameters from examples.*

Integration of additional visual cues As figure 3 shows, we plan to integrate other visual cues with stereo, motion and intensity data. In particular, we will include texture and color. Because texture boundaries usually depend on changes of material or sharp changes in surface orientation, they could be used to support the line processes in the depth and motion modules. For color the goal is to find boundaries that delineate regions of constant albedo (at a coarse resolution, since small surface markings should not be "seen" at this stage). As in the case of depth and motion, intensity edges play a critical role for these two additional visual modules. Hurlbert and Poggio (see [21]) have sketched a possible scheme for coupling albedo with intensity edges.

It is important to notice that the combination of several visual cues not only allows reinforcement of evidence for, say, a depth discontinuity, but also achieves a classification of an intensity edge in terms of its underlying physical cause: for instance, whether it is due to a shadow or a depth discontinuity. Clearly, psychophysics can give useful indications of which interactions are important in the human visual system.

Local versus global constraints on the line process The line process provides a means for imposing important physical constraints on the discontinuities such as: continuity, relative spatial isolation and possibly collinearity. These constraints are enforced by using appropriate cliques and associated *energy* values. However, in our experience with Markov Random Field models applied to real data, a problem has emerged with the use of the line process. In many cases the property of collinearity that can be enforced in this way remains too local: discontinuities tend to be too jagged and sometimes even broken when integration with intensity edges is *not* used. How can one enforce the property of continuity or simply collinearity over larger distances within the MRF framework? The basic idea that we have begun to explore is to have a higher-level MRF that consists of "features", such as straight lines of different orientations, with its prior probability distribution, coupled (bidirectionally) with the line process lattice (see figure 8).

Tolerance in registration When data from different cues are combined, say from intensity and from stereo, they must be registered. Spatial coincidence is the main constraint exploited here. In general, however, one cannot expect that discontinuities in depth and intensity will always have *exactly* the same location. Because of errors in the early vision processes, effects of filtering, photometric effects and so on, depth discontinuities may be offset by one or more pixels from intensity edges. To deal with this registration problem the cone ot influence might be useful, in which the intensity edges facilitate (or don't veto) the formation of depth discontinuities. The cone of influence size should be on the order of the line process neighborhood. In this way the line process constraints will ensure collinearity within the cone-of-influence. Again, important information will come from psychophysics: we expect to learn how alignment of, for instance, intensity edges with depth discontinuities affects human vision.

Learning parameters from examples A critical problem in using MRFs is the problem of parameter estimation. The performance of the scheme depends critically on the natural temperature of the field, the potentials associated with the clique configurations, the coupling between the lattices, and so on. Parameter estimation should provide estimates for these factors; possibly by learning from a set of examples.

Does integration influence early vision modules? In our computational approach to integration we have tacitly assumed that information flows from the early vision modules to the integration stage — the coupled MRF system — but not backwards. The output of say, stereo, is modified by the outputs of other modules at the level of the MRFs but the stereo process itself — the matching, for instance — is not affected. The decision to neglect feedback interactions, from the integration stage to the early processes, in the present version of our theory is mainly due to reasons of simplicity. Without modifying our scheme in an essential way, it is easy to incorporate backward effects from the integration stage by assuming that the whole process from early vision algorithms to the integration stage can be controlled by a higher–

order system taking into account higher–level goals and the available results. If recognition is the goal, for instance, the current results of the recognition operation on the integrated information can control which early processes to apply, where, and how (i.e. which parameters to use). In this case, one may hope to develop a useful theory of integration without worrying at first about the problem of feedback.

A different possibility is that interactions between the integration stage and the early vision modules are an essential part of any integration theory and cannot be neglected even in a first-order approximation. In an extreme case one might not be able to separate the integration stage usefully from the early vision modules and even the modules one from another.

In principle, this is possible. The algorithms for the early processes can be regarded in several cases as MRFs themselves (regularization algorithms are special cases of MRFs[2,23]). Thus our coupling schemes for integration can be extended to couple the early processes. In practice, we expect that parameter estimation may become a very serious problem once the early vision processes are tightly coupled.

Hardware implementations As discussed elsewhere[19,21] the coupled MRF models used here can be implemented efficiently in mixed digital and analog hybrid networks. It is interesting that, the interaction underlying coupling between fields is of the type of a multiplication, logical-and or veto operation. These operations have some intriguing possible implementations in terms of the properties of synapses.

While it is certainly possible to implement the same mixed deterministic and stochastic algorithms described here in, say, VLSI technologies, it is also interesting to explore approximative deterministic algorithms that may be simpler and more efficient. Marroquin[16] has provided an encouraging initial analysis along with estimates of convergence properties.

References

[1] H. Barrow and M. Tenenbaum. *Recovering Intrinsic Scene Characteristics from Images.* Academic Press, New York, 1978.

[2] M. Bertero, T. Poggio, and V. Torre. *Ill-Posed Problems in Early Vision.* A.I. Memo No. 924, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1986.

[3] Andrew Blake and Andrew Zisserman. *Visual Reconstruction.* The M.I.T. Press, Cambridge, MA, 1987.

[4] John F. Canny. *Finding Lines and Edges in Images.* Technical Report TM-720, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1983.

[5] Patrick Cavanagh. Reconstructing the third dimension: interactions between color, texture, motion, binocular disparity, and shape. *Computer Vision, Graphics, and Image Processing*, 37:171–195, 1987.

[6] Haluk Derin and Howard Elliott. Modeling and segmentation of noisy and textured images using gibbs random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(1):39–55, January 1987.

[7] Michael Drumheller. Connection Machine stereo matching. In *Proceedings of the AAAI*, pages 748–753, August 1986.

[8] Michael Drumheller and Tomaso Poggio. On parallel stereo. In *Proceedings of the IEEE Conference on Robotics and Automation*, San Francisco, CA, 1986.

[9] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, November 1984.

[10] W. Eric L. Grimson. *From Images to Surfaces.* M.I.T. Press, Cambridge, MA, 1981.

[11] W. Danny Hillis. *The Connection Machine.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1985.

[12] B. K. P. Horn and B. G. Schunk. D^termining optical flow. *Artificial Intelligence*, 17, 1981.

[13] James J. Little, Guy E. Blelloch, and Todd Cass. Parallel algorithms for computer vision on the Connection Machine. In *Proceedings of the International Conference on Computer Vision*, London, England, June 1987.

[14] James J. Little, Heinrich Bulthoff, and Tomaso Poggio. Parallel optical flow computation. In *Proceedings of the Image Understanding Workshop*, pages 417–431, Los Angeles, CA, February 1987.

[15] David Marr and Tomaso Poggio. A theory of human stereo vision. *Proc. R. Scoc. London B*, 204:301–328, 1979.

[16] Jose L. Marroquin. Deterministic bayesian estimation of markovian random fields with applications to computational vision. In *Proceedings of the International Conference on Computer Vision*, London, England, June 1987.

[17] Jose L. Marroquin. *Probabilistic Solution of Inverse Problems.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1985.

[18] Jose L. Marroquin. *Surface Reconstruction Preserving Discontinuities.* A.I. Memo No. 792, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, August 1984.

[19] Jose L. Marroquin, Sanjoy Mitter, and Tomaso Poggio. Probabilistic solution of ill-posed problems in computational vision. *J. Amer. Stat. Assoc.*, 82:76–89, 1987.

[20] David W. Murray and Bernard F. Buxton. Scene segmentation from visual motion using global optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(2):220–228, March 1987.

[21] Tomaso Poggio. *Integrating Vision Modules with Co[illegible] MRF's.* Working Paper No. 285, Artificial Intelligence L[illegible]ory, Massachusetts Institute of Technology, 1985.

[22] Tomaso Poggio and Edward B. Gamble Jr. In preperation.

[23] Tomaso Poggio, Harry Voorhees, and Alan Yuille. *Regularized Solution to Edge Detection.* A.I. Memo No. 833, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 1985.

[24] Demetri Terzopoulos. *Computing Visible Surface Representations.* A.I. Memo No. 800, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, March 1985.

[25] Demetri Terzopoulos. The role of constraints and discontinuities in visible-surface reconstruction. In *Proceedings of the IJCAI*, pages 1073–1077, August 1983.

[26] Harry Voorhees and T. Poggio. Detecting textons and texture boundaries in natural images. In *Proceedings of the International Conference on Computer Vision*, London, England, June 1987.

Thinking Machines Technical Report 86.14

Introduction to Data Level Parallelism

With Programming Examples
for the Connection Machine® System

April 1986

Contents

List of Figures

Chapter 1

Data Level Parallelism

1.1 Parallelism in the World Around Us

Whenever many things happen at once, parallelism is at work. It is at work for one of two reasons: either because someone is in a hurry or because it is the natural course of events. If, for example, many people are working at once to compose a song, it is because someone is in a hurry. Music is a naturally sequential process. Physical phenomena, on the other hand, are almost always parallel. The wind in a wind tunnel does not blow over one square centimeter of an automobile body at a time. It blows across the whole frame at once, showing the engineers how the flow in one section interacts with the flow in another. If we simulate the wind in parallel, the results come faster as a natural consequence. The parallelism is being utilized, but it is not being artificially imposed. Other examples of fundamentally parallel phenomena include vision processing, information retrieval, and many types of mathematical operations.

1.2 Parallelism in Computer Systems

The same two motivations, doing things in a hurry and doing things more naturally, also motivate computer architects. Until recently, those architects who are focused on greater speed have obtained it from faster circuitry. Making the electronics twice as fast, or the memory twice as big, has traditionally been a cost-effective way to double the performance of a single-processor computer system. But now these gains have become much harder to achieve. Limits to circuit speed have been reached. So designers who are solely focused on speed are now seeking to inject parallelism into their designs. If two computers of traditional architecture can operate in parallel, the overall speed of the system can double.

There is, however, another starting point for the design process. Computer architects can go back to the problems themselves and understand the parallelism that has been there all along. Having understood it, they can build a system that exploits it directly. The first benefit of this approach is simplicity. A computer that fits the problems it solves is easier to use and program than a computer that doesn't. And it is also faster. Systems that couple to the inherent structure of a problem mine a deeper vein of parallelism. For this reason, they can dramatically outperform systems whose superficial performance specifications seem superior. When parallelism is imposed on a problem, a speed-up of ten is considered good. When inherent parallelism is exploited, speed-ups of 1000 are commonplace.

Some applications benefit much more than others. While certain problems do not have a large amount of parallelism, there is a large and growing body of important problems that do. For these applications the method of designing the computer around the inherent parallelism of the problem is proving to be outstandingly valuable. This approach is called "data level parallelism." The remaining sections of this report describe data level parallelism and its application to three very different computing problems. The implementation examples use the Connection Machine system, the first data level parallel computer available on the commercial market. (See reference [8] for further discussion of the Connection Machine system)

1.3 Two Styles of Computer Parallelism

All computer programs consist of a sequence of instructions (the control sequence) and a sequence of data elements. Large programs have tens of thousands of instructions operating on tens of thousands, or even millions of data elements. Parallelism exists in both places. Many of the instructions in the control sequence are independent; they may in fact be executed in parallel by multiple processors. This approach is called "control level parallelism." On the other hand, large numbers of the data elements are also independent; operations on these data elements may be carried out in parallel by multiple processors. This approach, as mentioned in the previous section, is called "data level parallelism." Each approach has its strengths and limitations. In particular, data level parallelism works best on problems with large amounts of data. Small data structures generally do not have enough inherent parallelism at the data level. When the ratio of program to data is high, it is often more efficient to use control level parallelism. But control level parallelism requires the user to break up the program and then maintain control and synchronization of the pieces.

1.4 The Connection Machine Data Level Parallel Computer

The Connection Machine computer from Thinking Machines Corporation is the first system to implement data level parallelism in a general purpose way. Since the computer is designed around the structure of real world problems, the best way to understand the Connection Machine architecture is to follow its use in solving an actual problem. A VLSI simulation example will be used for that purpose. In VLSI simulation, the computer is used to verify a circuit design before it is released to be manufactured. The Connection Machine system provides a very direct way to perform this simulation. Each transistor in the circuit is simulated by an individual processor in the system. The chapters which follow explain three more examples in much greater detail.

1.4.1 Program Execution

Data level parallelism uses a single control sequence, or program, and executes it one step at a time, just as it is done on a traditional computer. The Connection Machine system utilizes a standard architecture front end computer for this purpose. All programs are stored on the front end machine. Its operating system supports program development, networking, and low speed I/O. The front end computer has access to all the memory in the system, albeit one data element at a time because it is a serial computer.

All Connection Machine program execution is controlled by the front end system. A Connection Machine program has two kinds of instructions in it: those that operate on one data element and those that operate on a whole data set at once. Any single-data-element instructions are executed directly by the front end; that is what it is good at. The important instructions, those that operate on the whole data set at once, are passed to the Connection Machine hardware for execution.

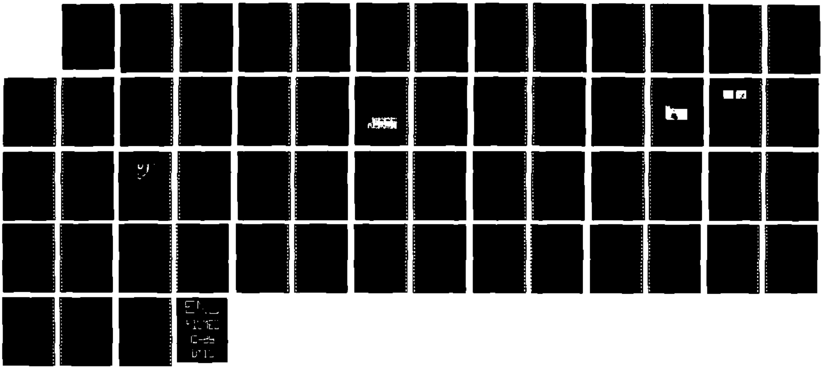
In the VLSI simulation example, the important instructions are the ones which tell each processor to step through its individual transistor simulation process. Each processor executes the same sequence of instructions, but applies them to its own data, the data that describes the voltage, current, conductance, and charge of its transistor at that time step of the simulation.

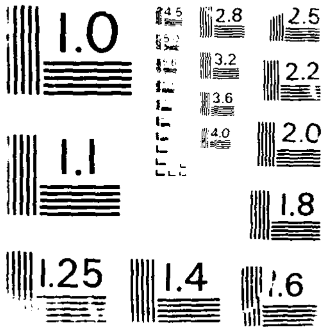
1.4.2 The Connection Machine Processors

In order to operate on the whole data set at once, the Connection Machine system has a distinct processor for each data element. The system implements a network of 65,536 individual computers, each with its own 4096 bits of memory. The data that describe the problem are stored in the individual processors' memories. During program execution, whenever the front end encounters an instruction which applies to all the data at once, it passes the instruction across an interface to the Connection Machine hardware. The instruction is broadcast to all 65,536 processors, which execute it in parallel.

Applications problems need not have exactly 65,536 data items. If there are fewer, the system temporarily switches off the processors that are not needed. If there are more problem elements, the Connection Machine hardware operates in virtual processor mode.

AD-A212 489 PARALLEL ALGORITHMS FOR COMPUTER VISION(U) 2/2
MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL
INTELLIGENCE LAB T POGGIO JAN 89 ETL-0529
UNCLASSIFIED DACA76-85-C-0010 F/G 12/1 NL




1.0
4.5
5.0
5.6
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

Each physical processor simulates multiple processors, each with a smaller memory. Virtual processing is a standard, and transparent, feature of the system. A Connection Machine system can easily support up to a million virtual processors. In general, a problem should have between ten thousand and a million data elements to be appropriate for the Connection Machine system.

1.4.3 Connection Machine I/O

Since the front end system has access to all Connection Machine memory, it can load data into that memory and read it back out again. For small amounts of data, this is a practical approach, but for large amounts it is too slow. A separate 500-megabit-per-second I/O bus is used instead. This bus is used for disk swapping, image transfer, and other operations which exceed the capacity of the front end.

1.5 Communications: The Key to Data Level Parallelism

Large numbers of individual processors are necessary for data level parallelism, but by themselves they are not enough. After all, there is more to a VLSI circuit than individual transistors. A circuit is made up of transistors connected by wires. Similarly, there is more to a Connection Machine system than just processors. A Connection Machine system is made up of processors interconnected by a massive inter-connection system called the router.

The router allows any processor to establish a link to any other processor. In the case of the VLSI simulation example, the links between processors exactly match the wiring pattern between the transistors. Each processor computes the state of an individual transistor and communicates that state to the other processors (transistors) it is connected to. All Connection Machine processors may send and receive messages simultaneously. The router has an overall capacity of three billion bits per second.

It is part of the reality of the world we live in that many things happen at once, in parallel. It is part of the beauty of the world we live in that these many things connect and interact in a variety of patterns. Looking at the whole problem at once requires a computer that combines the ability to operate in parallel with the ability to interconnect.

Since the structure of each problem is different, the interconnection pattern of the computer must be flexible. All linkages between Connection Machine processors are established in software. Therefore, the system can configure its processors in a rectangular grid for one problem and then into a semantic network for the next. Rings, trees, and butterflies are other commonly used topologies. The chapter on hardware describes router operation in greater detail.

1.6 Connection Machine Application Examples

Each of chapters 2, 3, and 4 describes a Connection Machine example in detail. First the algorithm is described, and then the actual program that implements this algorithm is presented and discussed. It is not necessary to study the program to appreciate the simplicity of the overall approach. Many readers will want to skip over these details. The third example, contour mapping, is quite sophisticated. Hence the program for this example is more complex than the two that precede it.

The initial Connection Machine languages are C* and *Lisp. C* is an extension of C and is appropriate for a wide range of general purpose applications. *Lisp is an extension of Lisp. Lisp, while less well known than C, is also an appropriate language for a wide variety of applications. Its primary use, however, has been in the field of artificial intelligence. Chapters 5 and 6 provide an introduction to these languages.

Chapter 2

Document Retrieval

There is too much to read. The written material for almost every discipline grows much faster than any one person can read it. Computers have not provided much relief to date. Now data level parallelism provides the computing power to implement significantly better solutions to the document retrieval problem. These solutions are more natural, so they require less user training. And they are much more accurate, so they give the user much greater confidence in the results.

2.1 Accessing Computer Data Bases

There are a number of systems today that provide on-line access to text information, but they perform poorly because they rely on a "keyword" mechanism for finding documents. The premise of a keyword system is that the relevance of a whole document can be determined by the presence or absence of a few individual words. Users enter one or more "keywords" or labels that they feel capture the sense of the information needed. All documents which either contain these words or have been indexed under these words are retrieved. Those that do not are ignored. Even with refinements, such as "Find all occurrences of 'New England Patriots' within ten words of 'Superbowl'," a keyword search generally tends to either find too many documents for the user to cope with, or too few for the user to find useful. It is a guessing game, with the user trying to imagine the most fruitful search terms.

Not all relevant documents contain the one particular word that the user chose, because writers use language differently. A search for documents containing the word "chips" may find five relevant documents, but miss ten others that were indexed under "integrated circuits" or "VLSI." Since the search yields only one third of the relevant documents, it would be considered to have a *recall* of 33%. Worse yet, the five relevant documents might be returned mixed into twenty other documents describing cookies or paint or other subjects where the word "chips" appears. Such a search would be considered to have a *precision* of 20%. Recent published testing has shown that recall results of as little as 20% are common with keyword based systems [1].

In short, keyword-based systems are very good at finding one or two relevant documents quickly. What they are poor at is producing a refined result with high recall and high precision. The Connection Machine document retrieval system provides a very powerful way for doing complete searches. It starts out using a keyword approach, but once the first relevant document is found, the whole approach changes. The user proceeds by simply pointing to one or more relevant documents and saying, in effect, "Find me all the documents in the database that are on the same subjects as this one." A document that has been identified as relevant by the user is referred to here as a "good document."

2.2 Algorithms for Document Retrieval

Data level parallelism makes massive document comparisons simple. The basic idea is this: a database of documents is stored in the Connection Machine system, one or more documents per processor. Once the first good document is found, it is used to form a search pattern. The search pattern contains all the content words of the document. The host machine broadcasts the words in the pattern to all the processors at once. Each processor checks to see if its document has the word. If it does, it increases the score for its document. When the entire pattern has been broadcast, the document that most closely matches the pattern will have the highest score, and can be presented first to the user.

The algorithm is simple to program because it takes advantage of innate characteristics of documents rather than programming tricks and second guessing. Every document is, in effect, a thesaurus of its subject matter. A high percentage of the synonyms of each topic appear because writers work to avoid repetition. In addition, variants of each word (such as plural, singular, and possessive forms), and semantically related terms also appear among the words in a particular article. Clearly not every synonym, variant, and related term will occur in a single article, but many terms will. Each reinforces the connection between the search pattern and the document. Spurious documents, on the other hand, will not be reinforced. The word "chip" will appear in an article about cookies, but "VLSI" and "integrated circuit" simply will not. In the overall scoring, truly useful documents are reliably separated from random matches. (See figures 2.1 and 2.2.)

2.3 Database Loading on the Connection Machine System

A document database may be constructed from sources of text such as wire services, electronic mail, and other electronic databases. For this description it is important to draw a

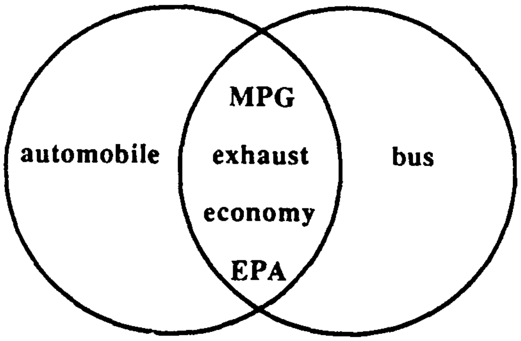


Figure 2.1: *Documents on the same subject have a high overlap of vocabulary.*

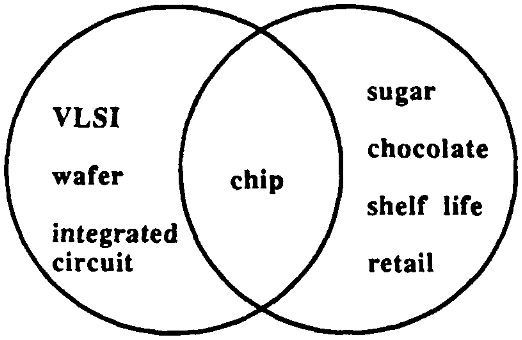


Figure 2.2: *Documents on different subjects have low overlap of vocabulary.*

distinction between *source documents* and *content kernels*. A *source document* contains the full actual text of a particular article, book, letter, or report, and is stored on the front-end's disk. A *content kernel* is a compressed form of the source document that encodes just the important words and phrases. It omits the commonplace words. Content kernels are stored in the memory of Connection Machine system.

The content kernel is produced automatically from the source document. First, the source document is processed by a Thinking Machines document indexer program that marks the most significant terms in the text. Next .hese terms are encoded into a bit-vector data structure, using a method called "surrogate coding." Surrogate coding, which is sometimes referred to as a "hash coding" method, allows the content kernel to be stored more compactly. It also speeds up the search process. In surrogate coding, each term in the content kernel is mapped into ten different bits in a 1024-bit vector. The ten selected bits in the vector are set to one to indicate the presence of the word in the document. In a content kernel of 30 terms, the process of surrogate coding ends up marking about a third of the bits as ones.

The source document in its original form is available for retrieval and presentation to the user when needed. The location of the original document on the system disk is stored with the content kernel.

Each segment of the content kernel is made up of the following fields:

`*score*` is used by the document lookup program to accumulate the ranking of each content kernel in the database according to how closely the content kernel matches the user's search pattern. Each time a match is found, `*score*` is updated.

`*document-id*` contains a reference to the original source document that this content kernel was derived from. When a content kernel is selected from the database lookup, the user is shown the source document referred to by this index.

`*kernel*` is a table of the surrogate-coded bit-vector encoding.

The necessary declarations for these fields are as follows. (In this chapter only, all of the code is presented twice, first in the *Lisp language and then in the C* language, to make it easy to compare the two languages. Because the characters * and ? may not appear in C* identifiers, such *Lisp names as `*score*` and `word-appears?` are rendered in C* simply as `score` and `word_appears`.)

```
;;; Declarations for the *Lisp version.

(defconstant table-size 1024)
(defconstant hash-size 10)
```

```
(*defvar *score*)
(*defvar *document-id*)
(*defvar *kernel*)


/* Declarations for the C* version. */

#define TABLE_SIZE 1024
#define HASH_SIZE 10

poly unsigned score, document_id;

poly bit kernel[TABLE_SIZE];
```

2.4 Document Lookup on the Connection Machine System

During the first stage of document lookup, the user lists a set of terms to be used to search the database, and receives back an ordered list of documents that contain all or some of those terms. The user then points to a document which is relevant, and from this document an overall *search pattern* of content-bearing words is assembled. The search pattern is simply a list of these words, with weights assigned to each word. The weight assigned to a word is inversely proportional to its frequency in the database (for example, "platinum" appears in the database less frequently than "gold," and therefore has a higher weight associated with it). This weighting mechanism ensures that uncommon words have more of an influence than common words over which content kernels get selected during the document lookup process.

Next, the search pattern is broadcast to all processors in the Connection Machine system. The same mechanism that is used to code each word in the content kernel as a series of bits is applied to the words in the search pattern. For each word in the search pattern a set of ten bit indices is broadcast. All content kernels that have these same ten bits set will have the weight of that word added into their `*score*` field. (It is possible that all ten bits for a word might happen to be set on account of other words even though that word doesn't really appear in the source document. Such an accident will result in a "false hit" on that word. However, for two reasons, this will not seriously affect the results of the lookup. First, the probability of a false hit is small: $(\frac{1}{3})^{10}$, or less than one in 50,000. Second, a false hit will be only one of many terms contributing to the score, and so will have only a small effect even when it does occur.)

The following code is used to broadcast *one* search pattern word to all the processors in the system, which check their content kernels and add the value of `weight` into their `*score*` if it contains the word. The word is represented by a list of ten bit locations (bit-locs).

```
;;; *Lisp code for testing the presence of a single word.

(*defun increment-score-if-word-appears (bit-locs word-weight)
  (*let ((word-appears? t!!))
    (dolist (bit bit-locs)
      (*set word-appears?
            (and!! word-appears?
                   (not!! (zerop!! (load-byte!! *kernel* (!! bit) (!! 1)))))))
       (*if word-appears?
             (*set *score* (+!! *score* (!! word-weight)))))


/* C* code for testing the presence of a single word. */

poly void increment_score_if_all_bits_set
        (mono unsigned word_bit_position[HASH_SIZE], mono int weight) {
    mono j;
    poly bit word_appears = 1;
    for (j = 0; j < HASH_SIZE; j++)
        word_appears &= kernel[word_bit_position[j]];
    if (word_appears)
        score += weight;
}
```

The main search program simply calls this routine once for each keyword in the keyword list.

2.5 Retrieving the Highest Scoring Documents

The code that follows is used to retrieve the `*document-id*` for each of the highest scoring content kernels in the database. The program returns a list of `*document-id*`s for the content kernels with the highest scores. The program first retrieves the `*document-id*` for the highest score, then the next highest score, etc., until a list of length `document-count` is retrieved. The `already-retrieved?` flag is set once a processor has had its `*document-id*` retrieved so it will not be retrieved again.

```
;;; *Lisp code for retrieving documents in order, highest score first.

(*defun retrieve-best-documents
  (let ((top-documents-list nil))
    (*let ((already-retrieved? nil))
      (dotimes (i document-count)
        (*when (not!! already-retrieved?)
          (*when (=!! *score* (*max *score*))
            (*let ((next-highest-document (*min (self-address!!))))
              (setq top-documents-list
                (append top-documents-list
                  (list (pref *document-id* next-highest-document))))
                (setf (pref already-retrieved? next-highest-document) t))))))
    top-documents-list))
```

```
/* C* code for retrieving documents in order, highest score first. */

poly void retrieve_best_documents
          (mono document_count, mono unsigned *document_id_array) {
    poly bit already_retrieved = 0;
    mono i:
    for (i = 0; i < document_count; i++) {
        if (!already_retrieved) {
            if (score == (><= score)) {
                processor *next_highest_document = (<>= this);
                document_id_array[i] = next_highest_document->document_id;
                next_highest_document->already_retrieved = 1;
            }
        }
    }
}
```

2.6 Timing and Performance

A production level version of the algorithms described above has been implemented and extensively tested on the Connection Machine system. Performance studies have been done on a database of 15,000 newswire articles, which constitute 40 megabytes of text. An automatic indexing system, selects the content kernels for each document. The content kernels are about one third of the original size of the text. Surrogate coding compresses the data by another factor of about two. In the system currently in use, the kernels are encoded into as many 1024-bit vectors as are needed at 30 terms per vector. For a long document several vectors are used; additional code, not shown above, is needed to chain the vectors together and combine the results.

Using this encoding, the Connection Machine system is able to retrieve the 2C nearest documents to a 200-word search pattern from a data base of 160 MBytes in about 50 milliseconds. (160 MBytes is equivalent to an entire year of news from a typical newswire.) In this time the Connect'on Machine system performs approximately 200 million operations for an effective execution speed of 6,000 Mips.

2.7 Summary and Implications

The program is brief because the algorithm is simple. The Connection Machine system is able to match the user's needs directly. It is powerful enough to carry out the algorithm in a straightforward way. The user wants to say to the database "All documents on the same subject as this one, line up in order here." That is exactly the service that the Connection Machine system provides for the user. It broadcasts the contents of the selected document to tens of thousands of processors at once. Each processor decides in parallel how similar its documents are. Then the most similar ones are sorted and presented to the user.

Even larger databases can use the same technique with two enhancements. The first enhancement is the use of a very high-speed paging disk, which allows larger numbers of content kernels to be swapped into the system for searching. The second enhancement is the use of cluster analysis. When the system has many documents on the same subject, it need not store all their content kernels individually. It can store one for the whole cluster, then retrieve the full set of related documents when needed. A single document may, of course, participate in more than one cluster. As the total database size grows, the size of the average cluster grows with it, making this a particularly appropriate technique for large scale databases. The addition of paging and clustering extends the algorithm described above to the 10-gigabyte range and beyond.

Chapter 3

Fluid Dynamics

Fluid flow simulation is a key problem in many technological applications. From the flow of air over an airplane wing to mixing in a combustion chamber, the problem is to predict the performance of a design without building and testing a physical model.

Until recently, fluid flow models were based almost exclusively on partial differential equations, typically the Navier-Stokes equations or approximations to them. These equations are not generally solvable by normal analytical methods. Numerical approximation techniques, such as finite difference methods and finite element methods, have been developed to solve these partial differential equations. All of these methods involve large numbers of floating point operations which require great amounts of fast memory. In addition, obstructions to the flow must usually be mathematically simple shapes.

Recent physics research has suggested that it is possible to make intrinsically discrete models of fluids. The fluids are made up of idealized molecules that move according to very simple rules, much simpler than the Navier-Stokes equations. The models are examples of cellular automata and are particularly well-suited to simulation on the Connection Machine. Cellular automata are systems composed of many cells, each cell having a small number of possible states. The states of all cells are simultaneously updated at each "tick" of a clock according to a simple set of rules that are applied to each cell. This approach involves only simple logical operations and does not require floating point arithmetic. It allows for all obstructions regardless of their shape. In addition, mathematical methods can be used to show that the results of such simulations agree with the results that would be obtained from the Navier-Stokes equations.

3.1 The Method of Discrete Simulation

Discrete simulation is used to model fluid flow on the Connection Machine system. The technique involves six key elements: particles, cells, time steps, states, obstacles, and interaction rules. *Particles* correspond to molecules of a fluid. A particle has a speed and a direction which determine how it moves. A *time step* is a "tick" of a clock that synchronizes the movement of particles. During each time step, particles move one cell in the direction that they are heading. A *cell* is a specific place in the overall region that is being observed. The region is completely filled with cells. Particles can move into and out of each cell during each time step. A *state* is a value assigned to each cell that indicates the number of particles within the cell, and in which directions they are heading. An *obstacle* is a set of special cells that obstruct the natural movement of particles. The *interaction rules* determine the movement of each particle when it shares a cell with one or more other particles. This movement is carried out by updating the state of the cells to reflect the new positions of the particles within the region.

A discrete simulation typically uses fixed cells. The cells never move or change during the simulation. Particles are completely in one cell during a time step, and move completely into the next cell (determined by the interaction rules) during the next time step. During each time step, every cell gathers data about particles heading in its direction from each of its neighboring cells. Based on the interaction rules, each cell determines the direction of its newly acquired particles and updates its own state.

A simulation designer can choose the cell topology and the interaction rules. The cell topology determines how many sides a cell has, and therefore, the directions by which particles may enter and exit. The simulation designer also determines the number of cells in the region being observed, and the average number of particles in each cell. Cellular automata theory provides the background for the simulation designer's decisions. It suggests that a simple cell topology, a huge number of cells and particles, and simple, local interaction rules are the most likely to be successful.

3.2 A Discrete Simulation of Fluid Flow

Thinking Machines is currently simulating fluid flow using a two-dimensional region that is divided into 3,000,000 hexagonal cells. Each cell is assigned to its own Connection Machine processor (using the virtual processor mechanism). The hexagonal mesh is a simple topology that gives the randomness that is required on a microscopic level to get correct results on the macroscopic level.

One of the fundamental reasons for computer simulation of fluid flow is to observe the behavior of a fluid as it flows past an obstacle. In the discrete model, obstacles are groups of cells that particles can not travel through. When a particle approaches an obstacle cell, it bounces off during the next time step. In order to observe the behavior of a fluid, tens of millions of microscopic particle interactions are simulated. Each individual particle's path through the cells and off of the obstacle cells appears almost random, just as in real fluids. However, when all of the particles' paths are considered, the overall behavior of the model is consistent with the way that real fluids behave. (See references [4,7,14] for further discussion of the use of cellular automata to model fluid flow.)

Individual particles can enter or exit through any of the six sides of each cell. A cell may contain a maximum of one particle heading in each of the six possible directions during a given time step (and so the total number of particles per cell per time step is anywhere from 0 to 6). A particle that has not collided with another particle during a time step will continue moving in the same direction during the next time step. (See figure 3.1.) When particles collide, a simple set of rules determines their new directions, conserving both momentum and the number of particles.

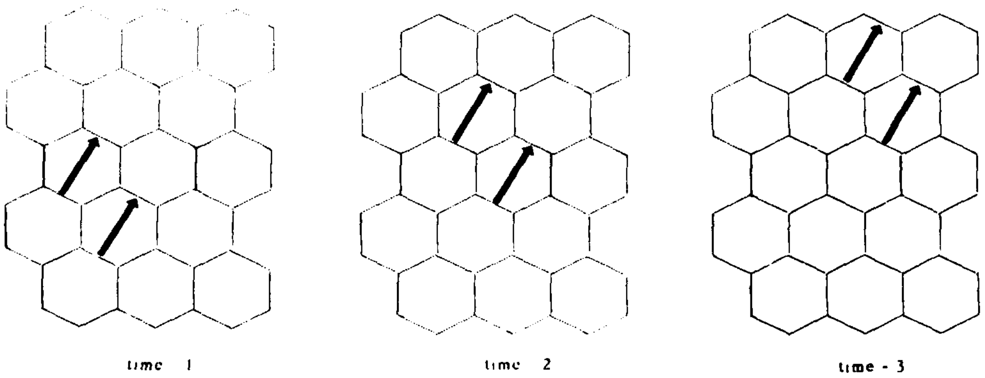


Figure 3.1: *Unless particles are obstructed by an obstacle, or collide into other particles, they continue in the same direction.*

At each time step, every cell updates its state by checking all of its adjoining cells, or neighbors, for particles that are heading in its direction. All cells then update their own states based on the information that they have gathered. In the model currently implemented, there are five situations that cause a particle to change directions: 2-way symmetric collisions, 3-way symmetric collisions, 3-way asymmetric collisions, 4-way symmetric collisions, and collisions with an obstacle cell. (See figure 3.2.)

Although the algorithm is implemented by modeling the individual movements and collisions of tens of millions of particles at each time step, the behavior of the fluid is observed by averaging the behavior of all of the particles in the entire region and by analyzing the

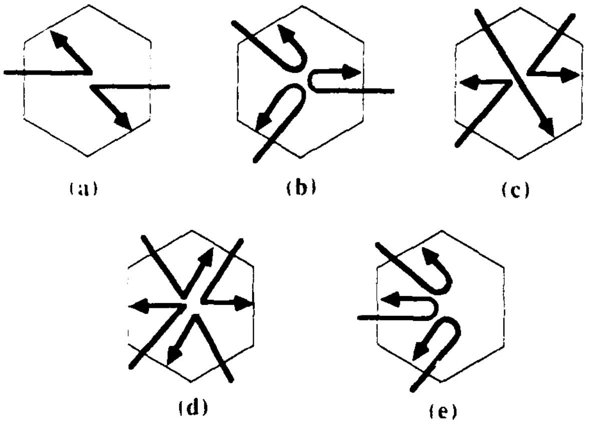


Figure 3.2: *Situations that cause particles to change directions.*
(a) *Two-way symmetric: two particles enter a cell from opposite sides. The particles exit through a different pair of opposite walls.*
(b) *Three-way symmetric: three particles enter a cell from non-adjacent sides. Each particle exits by the side through which it entered.*
(c) *Three-way asymmetric: three particles enter a cell, two of them from opposite sides. One particle passes through unobstructed; the other two particles behave as in a two-way symmetric.*
(d) *Four-way symmetric: four particles enter a cell, each particle's side is adjacent to only one other particle's side. Particles behave as in two two-way symmetric collisions (maximum of one particle exiting per side).*
(e) *Collisions with an obstacle cell: a particle always leaves an obstacle cell by the side through which it entered.*

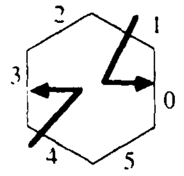
results over many time steps. In a typical simulation, macroscopic results are gathered by averaging particles together in groups of 20,000. Although each individual particle has only one speed and six possible directions, the average of 20,000 particles provides the full range of possible velocities.

3.3 Implementation on the Connection Machine System

There are two available ways for the Connection Machine system to implement the connections among the hexagonal cells. It can use the full router, setting up six connections for each site, one for each adjacent hexagon. Or it can use its grid, which connects four adjacent processors directly. The grid network was chosen for this implementation. It is very fast for small data transfers to nearby processors.

Of course, the grid cannot implement hexagonal connections directly. It connects to four adjacent processors, not six. Therefore, two of the six connections require two-step communication (i.e., up one and over one for the diagonal). The simulation program implements this two-step process. Each site can quickly learn the status of its six neighbors and can determine which ones contain particles that are moving in its direction.

Each cell has only 13 bits associated with it: six bits for incoming state (numbered 0–5), six bits for outgoing state (numbered 0–5), and one bit to indicate whether or not it is an obstacle. Each of the six incoming state and six outgoing state bits is dedicated to a particular direction. If a particle is entering or exiting through that direction, then the bit is set to 1, otherwise it is set to 0. (See figure 3.3.)



BITS	0	1	2	3	4	5
INCOMING	0	1	0	0	1	0
OUTGOING	1	0	0	1	0	0

Figure 3.3: ***Hexagonal cells with six incoming bits for particle direction and six outgoing bits for particle direction***

```
/* A cell state is represented by a six-bit unsigned integer,
   which can also be regarded as an array of six individual bits. */

typedef union STATE {unsigned:6 Val; unsigned:1 Bit[6];} state;

/* Each processor in the domain "grid" will contain a cell state
   (the outgoing state), another state (the incoming state) used
   for temporary purposes in the calculation, and a bit saying
   whether or not it is an obstacle cell. */

poly state outgoing_state, incoming_state;
poly unsigned:1 obstacle_cell;
```

```
/* The following declares the actual grid of processors. */

processor fluid_grid[ARRAY_X_SIZE][ARRAY_Y_SIZE];

/* Grid is the C pointer type that corresponds to the above array type. */

typedef processor (*grid)[ARRAY_Y_SIZE];
```

At each time step, instructions are broadcast that tell each cell how to gather data about particles heading in its direction. When the cells poll each of their six neighbors for information, they formulate their own 6-bit incoming state. For example, a cell would ask its East neighbor for its outgoing state bit number 3, and would place the answer in its own incoming state bit number 0. It would then ask its NorthEast neighbor for its outgoing state bit number 4 and would place the answer in its own incoming bit number 1. All cells, in parallel, check the state of all six of their neighboring cells. This extreme data level parallelism allows for a large amount of data to be collected in a small amount of time.

```
/* This code is executed within each processor.  Outgoing state
   bits from six neighbors are gathered and placed within the local
   incoming_state array.  Note the use of a C cast expression
   ((grid)this) to create a self-pointer that has a two-dimensional
   array type suitable for double indexing.  (This code actually is
   oversimplified in that it does not handle the boundary conditions
   for cells on the edge of the grid.  Handling these conditions is
   a bit tedious but conceptually straightforward.) */

poly void get_neighbors() {
    incoming_state.Bit[0] = ((grid)this)[ 1][ 0].outgoing_state.Bit[3];
    incoming_state.Bit[1] = ((grid)this)[ 0][ 1].outgoing_state.Bit[4];
    incoming_state.Bit[2] = ((grid)this)[-1][ 1].outgoing_state.Bit[5];
    incoming_state.Bit[3] = ((grid)this)[-1][ 0].outgoing_state.Bit[0];
    incoming_state.Bit[4] = ((grid)this)[ 0][-1].outgoing_state.Bit[1];
    incoming_state.Bit[5] = ((grid)this)[ 1][-1].outgoing_state.Bit[2];
}
```

Once each cell has determined which particles are entering (by collecting its incoming state), it updates its outgoing state to reflect the particle interactions. First, all cells that have their obstacle-bit turned on are instructed to set their outgoing state to be the same as their incoming state (since particles that hit an obstacle bounce back in the same direction).

Next, patterns are broadcast that correspond to each of the possible 6-bit incoming states, followed by the corresponding 6-bit outgoing state. Each cell compares its incoming state to the pattern being broadcast. When there is a match, the cell updates its outgoing state accordingly. For example, a cell with an incoming state of 011011 would then have an outgoing state of 110110 (refer to figure 3.2d).

```
/* The rule table is indexed by a six-bit incoming-state value
   and contains the corresponding outgoing-state values. */

state rule_table[64];

/* Calculate the new outgoing_state for all cells, based on the
   incoming_state and the obstacle_cell bit. */

poly void update_state {
    if (obstacle_cell)
         outgoing_state.Val = incoming_state.Val;
    else outgoing_state.Val = rule_table[incoming_state.Val].Val;
}
```

It is important to note that this trivial, non-computational, table look-up is the driving force of the whole simulation. The Connection Machine system has replaced all of the mathematical complexity of the Navier-Stokes equations with this small set of bit-comparison operations. The simulation is successful because the system can perform this operation on huge numbers of particles in very short amounts of time. It is an example of the Connection Machine system being easier to program because it supports a much simpler algorithm.

3.4 Interactive Interface

A typical "run" of a fluid flow simulation begins by allowing the user to make several choices. The user typically specifies the average number of particles per cell (density) and the average speed and direction of the particles (velocity). Technically this means that the entire region starts out with particles randomly distributed among the cells (based on the density) and moving in a certain overall direction (based on the average velocity). The user also selects or draws one or more obstacles and places them somewhere in the region being observed. All cells that are part of an obstacle have their obstacle bit set. As the simulation runs, new particles are randomly injected from the edges of the region in order to maintain the selected density and velocity. Once the model is running, each cell's state is continually updated, and average results for regions of cells are displayed.

```
/* This is the main computation loop.  At each time step, each
   cell fetches state from neighbors and updates its own state;
   then the results are displayed. */

poly void fluid_flow() {
    for (;;) {
        get_neighbors();
        update_state();
        display_state();
    }
}

/* Execution begins here. */

void start_fluid_flow() {
    /* Initialization. */
    initialize_rule_table();
    initialize_cell();
    /* Activate all processors in fluid_grid
       and then call the function fluid_flow. */
    [[][]fluid_grid].{ fluid_flow(); }
}
```

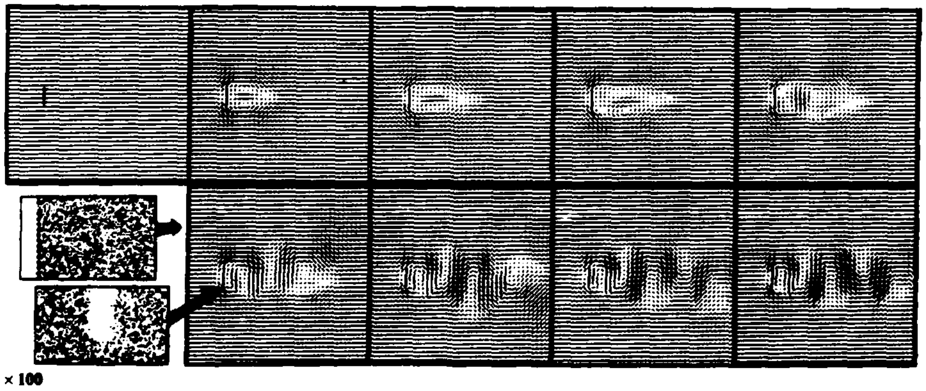


Figure 3.4: *The formation of a fluid flow phenomenon, called a "vortex street," as fluid flows from left to right past a flat plate.*

3.5 Timing and Performance

A production level version of the algorithm described in this chapter has been implemented and extensively tested on the Connection Machine system. The simulation operates on a 4000 × 4000 grid of cells, typically containing a total of 32 million particles. The Connection Machine system is able to perform one billion cell updates per second. Figure 3.4 shows several displays from a simulation of 100,000 time steps. Each time step includes approximately 70 logical operations per cell; the simulation therefore required a total of 100 trillion (10^{14}) logical operations. The complete simulation took less than 30 minutes. Current results are very competitive with state-of-the-art direct numerical simulations of the full Navier-Stokes equations.

3.6 Summary and Implications

In addition to providing very accurate simulation of fluid behavior, the Connection Machine method for simulating fluid flow allows scientists to continually interact with the model. Any of the user's original choices may be modified during a run of the simulation, without long delays for new results. Since particles are continually moving through the cells, a new density or average velocity may be established by adjusting the particles being randomly injected from the edges. When a new obstacle is added during a run, the obstacle bits in the appropriate cells are set, and those cells begin to reflect particles. Within less than a minute (a few thousand time steps), results based on the new selections become apparent in the displayed flow.

The algorithm for simulating fluid flow on the Connection Machine system is simple. It overcomes problems formerly associated with computer simulations of fluid flow by using a discrete simulation that takes advantage of the Connection Machine system's inherent data level parallelism. During each time step, every particle can move in the direction it is heading, every cell can evaluate its new particles based on collision rules, and every cell can update its state to reflect the direction of the particles it currently contains. The algorithm involves a small number of instructions executed over a large amount of data. Since the Connection Machine system is able to assign a processor to each data element, and to allow all processors to communicate simultaneously, it has provided the computational power required to provide the ideal solution to this applications need.

Chapter 4

Contour Maps from Stereo Images

Human beings have extremely sophisticated and well-developed visual capabilities, which scientists are just now beginning to understand. Since humans are very good at dealing with visual data, graphics and image processing provide an excellent opportunity for creative partnership between people and computers. An example of this partnership is the widespread use of graphical output for computer applications, such as scientific simulations. The computer does what it does best, computing the results and displaying them in a picture or a movie. Researchers do what they do best, using their sophisticated visual system to make qualitative judgements based on the visual information.

In many important computer applications, however, this partnership breaks down. When the flow of visual data is too large, the human visual system makes mistakes. Often this is simply because humans get tired and lose their concentration when faced with very large and monotonous streams of visual data, not because they are trying to extract information too subtle for current computer science to handle.

4.1 Analyzing Aerial Images by Computer

The analysis of detailed aerial images is an area where increased computer processing is highly desirable. Topographers would like to have the computer partially "digest" the visual data first, presenting only the essential properties of the images to the human user. In some cases, they would like to have the computer go even further, drawing abstract conclusions from raw visual data. Scientific progress in image processing and artificial intelligence has recently made this kind of information processing possible. However, conventional computers cannot keep up with the enormous flow of data that these applications present. Consequently, humans are still doing most of the work in these areas. The partnership has broken down because people are doing what the computer should be doing for them.

Data level parallelism is helping to redress this balance. It is ideally suited to the analysis of multiple images and the detection of subtle differences between them. In particular, it is allowing stereo vision algorithms to be applied to terrain analysis in very high volume applications. Stereo vision is the process by which humans are able to take in two slightly different images (from the two eyes) and use the small differences arising from the two different perspectives to determine the distances to the objects in the field of view. Using the same principle, the Connection Machine system is able to analyze two aerial images to determine the terrain elevation and to draw a contour map. Contrary to the apparent ease with which humans can perform this process, it is a subtle and difficult computational problem which no computer has yet solved perfectly. That is why humans are always involved to "coach" the process. The Connection Machine system, with its natural ability to handle large numbers of images and compare them in great detail, can help to drastically reduce the amount of work people must do in this area.

This chapter describes the underlying algorithms for stereo vision on a data level parallel computer, and shows some of the implementation on the Connection Machine system. Many detailed elements of an actual production system, such as straightening out misaligned images and displaying intermediate results, have been omitted in order to focus on the underlying algorithms. See references [2,3,5,11,12,13] for more information on machine vision and the stereo matching problem.

4.2 Seeing in Stereo

Images are very large, inherently parallel data structures. Therefore the processing of images is an application that is ideally suited for data level parallelism. An image is stored as an array of *picture elements*, or *pixels*. An image with 256 pixels in the vertical dimension and 256 in the horizontal dimension has a total of 65,536 data elements. More detailed images, with 1024 by 1024 pixels, have more than a million data elements. For black and white images, the value stored in each of the pixels is the intensity of light at that point, ranging from pure white through various shades of gray to pure black. (Pixels in color images contain information describing the hue and saturation as well as the brightness.) The contour mapping problem is one of extracting terrain *elevation* information from images that, upon first inspection, contain only information about terrain *brightness* at each pixel.

The term *stereo* means "dealing with three dimensions." *Stereo vision* is "the ability to see in three dimensions." Humans and many animals have the remarkable ability to take in two images, obtained from slightly different perspectives—one from each eye—and fuse them to perceive a three-dimensional world. The difference in perspective causes objects to appear in slightly different places in the two images. The amount of positional difference is related to the distance of the object from the viewer.

Because stereo vision occurs automatically in humans, we tend to be unconscious of the process. A simple demonstration serves as a reminder. Hold a pencil in front of a piece of paper and fix your gaze on the paper. Start to alternately close one eye and then the other, then slowly move the pencil toward your face. Keep the paper stationary and your gaze fixed on the paper while you move the pencil. The paper always seems to shift back and forth by the same small amount, but the closer the pencil moves to you, the more it jumps in position between the two views.

The two images used in a stereo vision system are called a "stereo pair." Figures 4.1 and 4.2 give an example. Figure 4.1 shows a model of some terrain, as seen from an oblique angle. Figure 4.2 shows a stereo pair obtained from directly above the terrain. Figure 4.2 can produce a vivid sensation of depth when observed with an appropriate stereo viewing apparatus.

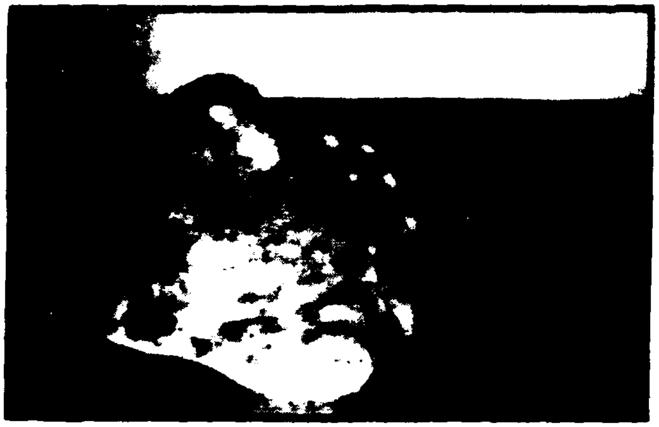


Figure 4.1: *An oblique view of a terrain model used in a demonstration of the contour mapping algorithm.*

4.3 Finding the Same Object in Both Images

Individual pixels within an image are not reliable indicators of objects. Two pixels, one in each image, can have the same brightness value without being part of the same object. Features larger than individual pixels must be found. The "edges" between areas of different intensities make up an effective set of such features. An edge is a line, usually a crooked line, along the boundary between two areas of the image that have different intensity. Instead of trying to match pixels based on their intensity, the algorithms match them based on the *shape of nearby edges.* The shape of edges is usually much more strongly related to

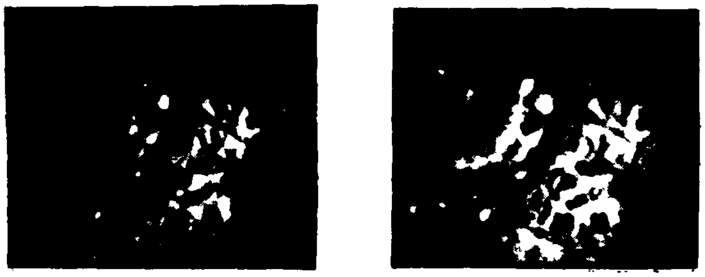


Figure 4.2: *A stereo pair of the terrain in Figure 4.1, obtained from directly above the terrain.*

distinct objects than the simple brightness value.

Figure 4.3 shows an example of edges. These edges were derived from the stereo pair in Figure 4.2.

The process of finding edges falls into the category of image computations called "local neighborhood operations." Individual pixels are classified based on characteristics of a group, or neighborhood, of nearby pixels. Edges are found by having each pixel determine whether the brightness of nearby pixels on one side of it is very different from the brightness of nearby pixels on the other side. This will be the case only for pixels that pass this test: *they must lie between two image regions that are similar within themselves but different from each other.* These *edge pixels* are detected by examining the local neighborhood of every pixel *in parallel*, and storing the ones that pass the test in an array. Typically, only 10 to 20 percent of the pixels in an image get classified as edge pixels.

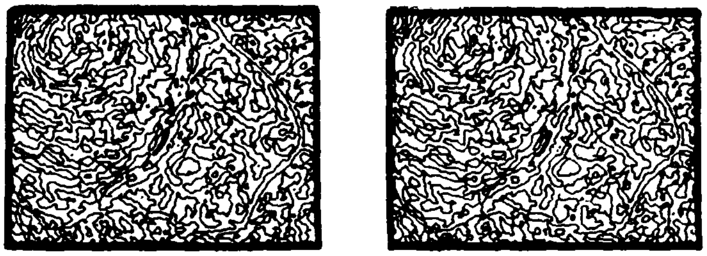


Figure 4.3: *An example of edges. These edges were derived from the stereo pair shown in Figure 4.2. They delineate the boundaries between areas of different intensity.*

4.4 Matching Edges

Even though edges are much more closely tied to objects than simple brightness values, there is still a great deal of work involved in deciding whether an edge in one image corresponds to a particular edge in the other image. Real images suffer from distortions due to several sources. Distortions include random fluctuations or "noise" introduced in the electronic imaging process, relative misalignment between the cameras, and irregular illumination. In addition to these effects, which tend to blur the distinction between edges that match and those that do not, there is a "bad luck" factor: an object or surface marking in one image very often just happens to look like several markings in the other image. For these reasons, the final choice of matches, and therefore the correct positional difference, is always somewhat ambiguous.

If the detection of edges were a perfect process, deciding which positional difference is best for each pixel would be simple. A local neighborhood of edges would align exactly at one relative shift and very little at all the others. Because of the imperfections described above, however, such a high level of precision is impossible. Every neighborhood of edges in one image matches to some extent with many neighborhoods in the other image. The competition is usually very close.

4.5 Measuring Alignment Quality

To resolve the competition, the Connection Machine algorithms hold one of the images stationary and "slide" the other one over it horizontally one pixel at a time. Each time the moving image is slid one more pixel's distance, all the stationary pixels compare themselves to the pixels to which they now correspond in the slid image. They record the presence or absence of an edge alignment in a table in their own memory. Typically, the maximum shift between two images is 30 pixels, so a table of 30 alignment matches is created in the memory of each stationary pixel's processor.

This sliding procedure, using the edges from Figure 4.3, is illustrated in Figure 4.4. Each of the 16 images shows an alignment table entry for each pixel. Black pixels indicate positive alignment table entries, i.e., "match-ups" between the stationary and the sliding images. For example, the 7th image shows alignment-table-slot 7 in each pixel. Thus every black pixel in image 7 corresponds to a match-up between stationary and sliding edges when the relative shift was 7 pixels.

The resulting alignment tables generally show several spurious matches, but also one or two solid ones where the local neighborhood of edges lined up very tightly. When this happens at a pixel, it is a signal that the correct shift (the correct positional difference) for that pixel has been found.

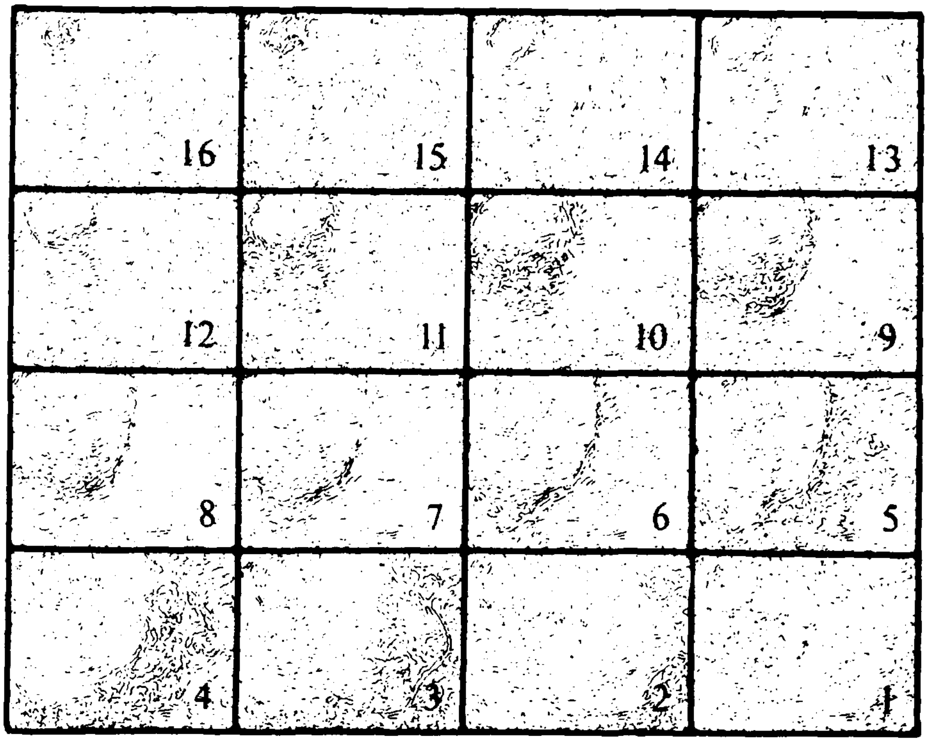


Figure 4.4: *An illustration of the sliding process. Each of these images shows the contents of an alignment-table-slot in each pixel. The Nth image shows slot N in every pixel's alignment table. The dark areas are regions of good alignment, i.e., areas where the same alignment-table-slot is filled in many pixels.*

As in the edge detection process, the alignment quality of every shift position in the alignment table is measured by a local neighborhood operation. In this case, the operation is the following: for each shift position, each pixel processor counts and records the number of matching edge pixels in a small neighborhood around itself. This count or "score" will be high for pixels whose nearby edges are tightly aligned with the edges in the other image *at the same position but displaced by the shift.*

The best shift for a given pixel is determined by comparing the alignment scores at every position in its alignment table. ***The shift that has the highest score is chosen as the correct shift for the pixel.*** This process takes place in parallel for all pixels; in this way a shift is determined for each pixel.

Areas of tight alignment are clearly visible in Figure 4.4. For example, the small shifts (1 through 4) are tightly aligned over low terrain (refer to Figure 4.1), and the large shifts (13 through 16) are tightly aligned over high terrain. Match-ups in these areas will get high alignment scores because they lie amidst many other match-ups.

4.6 Drawing Contour Maps

The processing described so far yields the shift (or elevation) for every pixel that is part of an edge. These pixels form a "web" of heights that approximates the shape of the terrain, but is not yet smooth and continuous. It is full of holes (where non-edge pixels were) which must be filled in by interpolation.

Interpolation is accomplished by another local neighborhood operation. Each pixel that is not on the web takes on a new elevation which is the average elevation of the pixels in a small neighborhood around it. The neighborhood includes the four pixels above, below, to the left and to the right of the pixel. The pixels that make up the web maintain their original elevations; only the pixels in the holes change their values. This process is repeated or "iterated" a few hundred times.

Pixels that lie in the middle of holes in the web have zero elevation. Therefore, when they become the average of their neighbors, which also have zero elevation, their elevation does not change. However, pixels that lie near the edges of holes in the web have neighbors whose elevation is nonzero. Therefore, when they become the average of their neighbors, they jump to a nonzero elevation. On the next iteration, these new nonzero pixels influence their neighbors, in turn creating new nonzero elevations. Gradually, after a few hundred iterations, the pixels on the web—which remain unchanged throughout the process—"spread" their elevations across the holes in the web, filling it in to create a smooth, continuous surface from which a contour map may be drawn. An example of a contour map is shown in Figure 4.5.

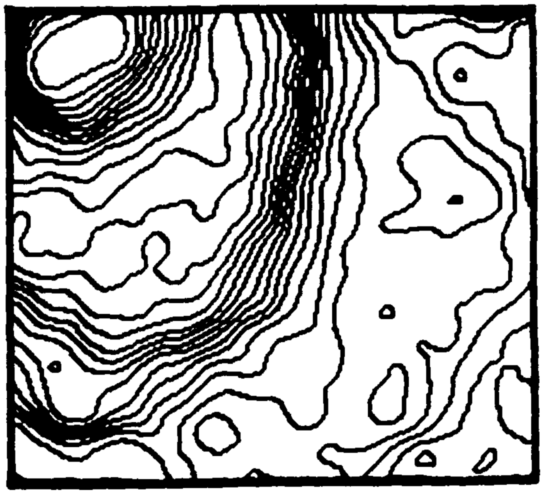


Figure 4.5: *A contour map of the terrain model shown in Figures 4.1 and 2, computed on the Connection Machine system.*

4.7 Finding Edges on the Connection Machine System

A pixel is classified as an edge pixel if it lies between two image regions that are similar within themselves but different from each other. This is the program that performs the edge classification operation.

```
(*defun find-edges-between-left-and-right!! (brightness-pvar threshold)
  (*let* ((average-brightness-on-the-left
            (/!! (+!! (pref-grid-relative!! brightness-pvar (!! -1) (!! -1))
                      (pref-grid-relative!! brightness-pvar (!! -1) (!! 0))
                      (pref-grid-relative!! brightness-pvar (!! -1) (!! 1)))
                 (!! 3.0)))
          (average-brightness-on-the-right
            (/!! (+!! (pref-grid-relative!! brightness-pvar (!! 1) (!! -1))
                      (pref-grid-relative!! brightness-pvar (!! 1) (!! 0))
                      (pref-grid-relative!! brightness-pvar (!! 1) (!! 1)))
                 (!! 3.0)))
          (average-brightness-overall
            (/!! (+!! average-brightness-on-the-left
                      average-brightness-on-the-right)
                 (!! 2.0))))
    (if!! (>!! (absolute-value!! (-!! average-brightness-on-the-left
                                      average-brightness-on-the-right))
               (*!! (!! threshold) average-brightness--overall))
          (!! 1)
          (!! 0))))
```

The preceding program sequence calculates the average brightness in a small region to the left (i.e., with relative x-coordinate −1, and relative y-coordinates −1, 0, and 1) and the average brightness in a small region on the right side (with relative x-coordinate 1) of each pixel. If, at any particular pixel, the difference between these averages is greater than the specified threshold, then the pixel is marked with a one, meaning that it is an edge pixel. Otherwise it is marked with a 0. The threshold is multiplied by the overall average brightness, a process called "normalization." With normalization, the threshold adapts to the image, becoming small in regions where the image is generally dark, and large where the image is generally bright.

Since this program compares regions on the left and right sides of a pixel, it works only for edges that are more or less vertical. It is easy to write a program that finds horizontal edges by having it compare small regions on the top and bottom of a pixel, in the same way that this program compares regions on the left and right. The same could be done edges in both diagonal directions. The four programs may then be combined to find *all* edges in the following way:

```
(*defun find-all-edges!! (brightness-pvar threshold)
  (if!! (or!! (=! (!! 1) (find-edges-between-left-and-right!!
                            brightness-pvar threshold))
              (=! (!! 1) (find-edges-between-above-and-below!!
                            brightness-pvar threshold))
              (=! (!! 1) (find-edges-between-upper-left-and-lower-right!!
                            brightness-pvar threshold))
              (=! (!! 1) (find-edges-between-lower-left-and-upper-right!!
                            brightness-pvar threshold)))
         (!! 1)
         (!! 0)))
```

4.8 Matching Edges on the Connection Machine System

The following program sequence implements the sliding procedure described above. One of the edge images is held stationary and the other edge image is moved across it horizontally, one pixel at a time. At each relative shift (1, 2, ..., 30), each processor records whether an edge match has been found in the sliding image. This information is stored in a pvar that represents one of the alignment tables discussed above. All of the alignment tables are stored in the Connection Machine memory at the same time.

```
(defvar *array-of-pvars-holding-matches-at-each-shift* (make-array 30))
  ;;; This is just a regular Lisp array, but each element of this
  ;;; array will be a pvar.  Notice that we'll try to find positional
  ;;; differences of up to 30 pixels.  (Note: each one of the pvars
  ;;; in this array will hold an "alignment-table-slot" for every pixel,
  ;;; as discussed in the text).

(*defun fillup-pvars-wherever-edges-align (left-edges right-edges)
  ;; This program records the edge-pixel match-ups at every shift;
  ;; that is, this program creates "match-up images," as shown in
  ;; Figure 4.4.
  (dotimes (i 30)
    (aset (if!! (=!! left-edges
                     (pref-grid-relative!! right-edges (!! i) (!! 0))
                     )      ; ^This PREF-GRID-RELATIVE!! accomplishes
                (!! 1)      ; the "sliding" process.
                (!! 0))
          *array-of-pvars-holding-matches-at-each-shift*
          i)))
```

The next step in the process is to decide at each pixel position which shift produced the best match-up. Most locations will contain a somewhat random pattern of match-up pixels. However, at some locations, the local neighborhood of match-ups will be very dense and regular, indicating that the shift responsible for that match-up image is probably the correct shift for that neighborhood.

The following *Lisp program measures the density or alignment quality of every neighborhood. It does so by counting the number of 1's (match-ups) in a square around each pixel. The counting process is accomplished in parallel, for all pixels at once, on the Connection Machine system.

```
;;; The square for each pixel is to be centered on that pixel.
;;; Because a DOTIMES loop always produces values starting at zero,
;;; it is necessary to subtract one-half the width of the square
;;; from the loop variable in order to get relative indexes that
;;; are centered on zero.

(*defun add-up-all-pixels-in-a-square (pvar width-of-square)
  (let ((one-half-the-square-width (/ width-of-square 2)))
    (*let ((total (!! 0)))
      (dotimes (relative-x width-of-square)
        (dotimes (relative-y width-of-square)
          (*set total
                (+!! total
                     (pref-grid-relative!!
                        pvar
                        (- relative-x one-half-the-square-width)
                        (- relative-y one-half-the-square-width))))))
      total)))
```

At this point, it is a simple matter to record the alignment quality or score for every pixel.

```
(defvar *array-of-pvars-holding-scores-at-each-shift* (make-array 30))
  ;;; Another Lisp array holding *Lisp pvars.
```

The next step is to fill all the elements of the Lisp array with *Lisp pvars. The Nth element of the Lisp array holds a pvar containing the scores, or alignment qualities, of all the matches that occurred when the edge images were shifted by N pixels relative to each other. (Note that this program records scores only at locations where match-ups occurred. Other locations have no score, which reflects our original intention of matching *edges*, not the holes between them.)

```
(*defun fillup-pvars-with-match-scores (width-of-square)
  ;; WIDTH-OF-SQUARE will typically be 21.
  (dotimes (i 30)
    (*let ((sum-of-all-nearby-pixels
             (add-up-all-pixels-in-a-square
               (aref *array-of-pvars-holding-matches-at-each-shift* i)
               width-of-square)))
       (*if (=!! (aref *array-of-pvars-holding-matches-at-each-shift* i)
                 (!! 1)) ;;; Record a score wherever there was a match-up.
            (*set sum-of-all-nearby-pixels
                  *array-of-pvars-holding-scores-at-each-shift*
                  i)))))
```

Now that the score for every match-up has been recorded, there is only one more step required to establish which of the match-ups is correct. The following *Lisp program loops through all the shifts, keeping track of the best score at each pixel. The shift that produced the best score at each pixel is recorded as the "winning shift."

```
;;; This function computes the web of known shifts.  Recall that
;;; the shift at each pixel corresponds directly to the elevation.

(*defun find-the-shifts-of-the-highest-scoring-matches ()
  (*let ((best-scores (!! 0))
         (winning-shifts (!! 0)))
     ;; The following DOTIMES loop makes sure that each
     ;; pixel in the BEST-SCORES pvar contains the maximum
     ;; score found at any shift.
     (dotimes (i 30)
       (*if (>!! (aref *array-of-pvars-holding-scores-at-each-shift* i)
                 best-scores)
            (*set best-scores
                  (aref *array-of-pvars-holding-scores-at-each-shift* i))))
     ;; The following DOTIMES loop records a "winning"
     ;; shift at every pixel whose score is the best.
     (dotimes (i 30)
       (*if (=!! (aref *array-of-pvars-holding-scores-at-each-shift* i)
                 best-scores)
            (*set winning-shifts (!! (1+ i)))))
     winning-shifts))
```

4.9 Drawing Contours on the Connection Machine System

A contour map cannot be constructed without a smooth, continuous surface on which to draw the lines. All of the processing so far has produces a web of known elevations (returned by the last *Lisp function above). Interpolation across the holes in the web produces a continuous surface.

```
(*defun fill-in-web-holes (web-of-known-elevations times-to-repeat)
  ;; Each time through the loop, every pixel not on the web (i.e.,
  ;; every pixel that is not zero to begin with) takes on the
  ;; average elevation of its four neighbors.  Therefore, the web
  ;; pixels gradually "spread" their elevations across the holes,
  ;; while they themselves remain unchanged.
  (dotimes (i times-to-repeat)
    (*let ((not-fixed (zerop!! web-of-known-elevations)))
      (*if not-fixed
         (*set web-of-known-elevations
               (/!! (+!! (pref-grid-relative!!
                           web-of-known-elevations
                           (!! 1) (!! 0))          ;Neighbor to the right
                         (pref-grid-relative!!
                           web-of-known-elevations
                           (!! 0) (!! 1))          ;Neighbor above
                         (pref-grid-relative!!
                           web-of-known-elevations
                           (!! -1) (!! 0))         ;Neighbor to the left
                         (pref-grid-relative!!
                           web-of-known-elevations
                           (!! 0) (!! -1)))        ;Neighbor below
                   (!! 4))))))
   web-of-known-elevations) ;;; this is now a more or less smooth surface.
```

The following code takes the smoothed-out web and constructs a contour map in the form of a plane of black-and-white pixels suitable for display on a graphics device.

```
(*defun draw-contour-map (number-of-contour-lines
                          pvar-of-smooth-continuous-elevations)
  ;; The idea is to divide the whole range of elevations into
  ;; a number of intervals, then to draw a contour line at every
  ;; interval.
  (let* ((max-elevation (*max pvar-of-smooth-continuous-elevations))
         (min-elevation (*min pvar-of-smooth-continuous-elevations))
         (range-of-elevations (- max-elevation min-elevation))
         (contour-line-interval (/ range-of-elevations
                                   number-of-contour-lines)))
```

```
 ;; Now the variable CONTOUR-LINE-INTERVAL tells us how many
 ;; elevations, or shifts, to skip between contour lines.
(if!! (zerop!!
        (mod!! (-!! pvar-of-smooth-continuous-elevations
                   (!! min-elevation))
               (!! contour-line-interval)))
      (!! 1)    ;; This IF!! draws all the elevation contours
      (!! 0)))) ;; at once, returning a bit map suitable for
                ;; for immediate display.
```

4.10 Timing and Performance

A production level version of the contour mapping algorithm described in this chapter has been implemented and extensively tested on the Connection Machine system. Parameters such as the size of the images and the range of positional differences ("shifts") are variable, depending on the application. A typical program run processes images containing 512 × 512 (262,144) pixels, while allowing for positional differences from 0 to 30 pixels. In such a mode, the Connection Machine system performs approximately two billion (2×10^9) operations during the most time-consuming phase of the algorithm, the so-called "inner loop," in which the match-ups are detected and their alignment quality is measured. This inner loop is executed in less than two seconds.

4.11 Summary and Implications

Contour mapping using stereo vision is an example of an image processing application that is sophisticated and computationally expensive. The Connection Machine system, because it readily accommodates itself to the inherently parallel structure of image data, made it easy to conceptualize and to program the contour mapping algorithm. The simplicity and brevity of the programs shown above is evidence of this natural fit.

The raw speed of the Connection Machine system is as valuable as its architecture. The system can extract elevation information from large amounts of visual data at very high rates. This speed allows scientists and engineers who are developing new techniques in computer vision to try their ideas "on the fly." A short turnaround time for experimenting with new ideas is essential for the rapid development of the field of computer vision. The effects of various program modifications are realized almost instantaneously. The system's computational power is a valuable aid in the design and implementation of sophisticated algorithms.

Chapter 5

The C* Programming Language

C* (pronounced *see star*) is a simple extension to the C programming language [6,10] that exploits the power of the Connection Machine architecture. C* is (almost) a strict extension of C; any valid C program, if it avoids the use of a small number of C* reserved words, is also a valid C* program. A few new features of the language serve to indicate where data is stored and which operations are executed in parallel in the Connection Machine network.

5.1 C* Extensions

In order to indicate whether a variable is located on the host or in the Connection Machine memory, two storage class identifiers mono and poly have been included in C*.

```
mono int x;      /* x resides in the host memory */
poly int y;      /* y resides in the Connection Machine memory */
```

The modifier poly declares variables present in all processors.

The majority of parallel code is standard C code. Parallel functions are simply distinguished by the identifier poly. It is a mark of the general-purpose nature of the Connection Machine architecture that the full C language is available for programming the processors of the Connection Machine system. Likewise, it is a mark of the simplicity of the architecture that the C language suffices for this task. In fact, no new language features need to be introduced in order to perform parallel control flow, interprocessor communication, and memory allocation. The real power of C* comes from the natural parallelization of familiar constructs of C.

5.1.1 Parallel Control Flow

Inside of a parallel function, the normal C control-flow statements, such as if and while, work as expected. This is perhaps unexpected to someone experienced with other parallel languages. For example, an if statement may have a conditional expression whose value is different in different processors:

```
poly salary;
...
if (salary <= 0)
    salary = fixup_salary();
```

It would clearly be an error for *all* processors to make the call to fixup_salary. The way C* handles such a statement is to reduce the active set of processors, by temporarily inactivating all those whose salary variables are positive. The body of the if statement is run, and then the original active set is restored. Such conditional statements can be nested to any degree.

The while statement can also operate in parallel. At each evaluation of the loop's conditional expression, more processors can drop out of the active set; they stay inactive until the loop is finished. Finally, when all processors are finished with the loop, the statement is done, and the original active set is restored. For example:

```
poly resumes_to_read;
...
while (resumes_to_read > 0) {
    /* Read ten resumes at a time. */
    resumes_to_read -= 10;
    ...
}
```

In this case, all processors with resumes_to_read between 1 and 10 execute the loop body exactly once.

All other standard C control constructs are handled in similar ways in C*; even *goto* is accommodated. The program behaves as if the standard C code were running separately in each processor, with processors that are doing the same thing doing it at the same time.

5.1.2 The Selection Statement

In order to execute code in a selected set of processors, an additional statement called the *selection statement* is included in C*. Selection statements may be used within any C* function. The selection statement has the form:

```
[selector].statement
```

The selector indicates a set of processors. These are activated, and the `statement` is executed within those processors. For example, given the following declaration,

```
processor managers[100];
```

the following statement

```
[[100]managers].{ salary *= 1.06; }
```

or, more simply,

```
[[]managers].{ salary *= 1.06; }
```

selects all 100 of the `managers`, and gives them a six percent raise. The code:

```
[[50]managers].{ salary *= 1.11; }
```

gives the first 50 an eleven percent raise, while this:

```
[managers[0],managers[2]].{ salary -= 1000; }
```

singles out the first and third managers for a pay cut. (More complicated forms of selection are also available.)

5.1.3 Computation of Parallel Expressions

C* extends the meaning of C expressions to parallel computations by means of two simple rules. The first rule says that if a single value (typically of storage class `mono`) is combined with a parallel value (of class `poly`), the single value is first replicated to produce a `poly` value. (In hardware terms, the single value is *broadcast* to all relevant processors.) For example, in the expression `(salary > 20000)`, the single value 20000 is replicated to match the parallel variable `salary`. This rule is an addition to the rules of "usual conversions" in plain C.

The second rule says that an operation on a parallel value (or values) must be processed *as if* only a single operation were executed at a time, in some serial order. In the expression `(salary > 20000)` it is *as if* we took first one `salary` value and compared it to 20000, then another, and so on, doing the comparisons one at a time.

Fortunately, we can analyze the > operation and determine that doing all the comparisons at once will produce the same result, because doing so will not affect the outcome. This is hardly surprising, and it is exactly the effect we want anyway, so why do we have the "as if serial" rule at all? It is because *some* operators *do* have side effects: assignment operators. Consider the expression

```
total_payroll += salary;
```

Now `total_payroll` is a single value (what in C is called an *lvalue*, because it occurs on the left side of an assignment). By the first rule it is replicated. We then have many assignments to perform, one for each value in the parallel value `salary`:

```
total_payroll += salary_1;
total_payroll += salary_2;
total_payroll += salary_3;
          .
          .
          .
```

The second rule guarantees that the program behaves *as if* a!! of these assignments were performed in some serial order. *Which* order does not matter; the result is the same. The point is that if these assignments were executed in parallel some updates might be lost; but C* guarantees that *all* the `salary` values will be correctly added into `tɔtal_payroll`. (Doing this efficiently is handled by the C* implementor.)

A C assignment operator may be used as a unary operator in C* to reduce a parallel value to a single result that may be further operated upon. For example,

```
(+= salary)
```

adds up the salaries for all persons for which processors are active, and

```
(+= salary)/(+= ((poly) 1)))
```

computes the average of all salaries because the expression

```
((poly) 1)
```

makes a 1 for every active processor and

```
(+= ((poly) 1)))
```

adds up all the 1's, thereby counting all the active processors.

In C*, "<>" is the "minimum" operator and "><" is the "maximum" operator. The expression "`a >< b`" means the same as "`(a > b) ? a : b`". The assignment operators `<>=` and `><=` are also defined: "`a <>= b`" assigns `b` to `a` if `b` is less than `a`. The expression `(><= salary)` finds the largest salary, and `(<>= salary)` finds the smallest salary.

5.1.4 Data Movement

C* has no language extensions to handle data movement or interprocessor communication *per se*. Instead, the normal C operations are used; the Connection Machine architecture allows random access to the desired datum, wherever it is in the system.

Within the code of a poly function, the keyword this is a C* reserved word whose value is a pointer to the currently executing processor. This value is sometimes called the *self-pointer*. If many processors are executing, each will have its own self-pointer. References to the processor's variables implicitly refer to the self-pointer: saying salary is the same as saying this->salary. Explicit references to this are useful for accessing the memory of neighboring processors through indexing.

The key point is that any processor may contain a pointer to data in the memory of any other processor, and access through that pointer is supported by the Connection Machine router. All interprocessor communication can therefore be expressed in C* merely by the usual explicit and implicit pointer indirection mechanisms. For example, to increment a neighbor's salary field, and then decrement one's own based on the result, the following code might be used:

```
this[1].salary += 1000;
salary -= this[1].salary * .10;
```

Similar expressions can also be used to broadcast data throughout the system, to transfer data between the host and Connection Machine processing network, or to collect data from many sources into one location.

5.2 Summary

The C* language is a version of the standard C language suitable for programming the Connection Machine system. Because of the simplicity and power of the Connection Machine architecture, C* itself is a simple yet powerful extension of C. The Connection Machine memory is treated as a large section of host-accessible memory with active objects stored in it. Because standard C is already excellent at manipulating structures, pointers, and the like, relatively few new language features are needed to deal with the Connection Machine architecture. All the familiar C language constructs acquire the power of parallelism easily and naturally.

Chapter 6

The *Lisp Programming Language

*Lisp (pronounced *star lisp*), is an extension of Common Lisp [9], a standard dialect of Lisp that is found on a variety of computer systems. Lisp has many features that are common to most programming languages, but its unusual structure and syntax make the programs a bit difficult to read for someone who has mainly had experience with block structured languages such as FORTRAN or C.

This chapter covers both Lisp and *Lisp in sufficient depth to make it possible to understand the program examples in this book. See references [9,15,16] for a deeper understanding of the Lisp language and its structure.

6.1 Fundamentals of Lisp

What most people remember about Lisp is that it uses lots of parentheses. And it is true—Lisp does. But it is not necessary to understand the full implications of the parentheses to understand the sample programs. Roughly, in a Lisp expression the first thing that comes after the open parenthesis is the function name, and after that are the arguments. So `(+ 7 A)` would call the function `+`, which adds 7 and the value of the variable `A`, and returns the result.

Lisp function calls can be nested as they can in other languages. For example:

```
(* 5 (+ 1 2 3))
```

would first add together 1, 2, and 3, and then multiply the result by 5, giving 30.

Most Lisp programs are indented to help reveal their structure and to show how many levels deep parentheses have been nested. Expert Lisp programmers keep their code properly indented, and rely on the indentation as much as the parentheses when reading code.

6.1.1 Lisp Functions

Functions are the program building blocks of Lisp. Unlike many other programming languages, Lisp does not have a main program followed by a series of functions. In Lisp everything is a function, and programs are executed by invoking those functions from an interactive Lisp interpreter.

The Lisp function-defining operation is called DEFUN. The first argument to DEFUN is the name of the function that is being defined, the second a list of its arguments; these are followed by the operations to be performed. For example:

```
(defun add-three (x) (+ x 3))
```

defines a function named add-three that takes one argument named x, and the operation that is performed by the function is (+ x 3).

6.1.2 Variables

It is not necessary in Lisp to predefine variables, but it is often done for clarity. The mechanism is straightforward:

```
(defvar a 25)
```

defines a variable named a with an initial value of 25. Variables defined with defvar are global variables that can be accessed by any function at any time.

Temporary variables are defined in Lisp with the let operation, which takes a list of variable-value pairs, and is followed by a sequence of operations to be performed. For example,

```
(let ((temporary 25)
      (x 49))
  (print (+ temporary x))
  (print (* temporary x)))
```

allocates two temporary variables temporary and x, assigns them the values 25 and 49 respectively, prints their sum and product, and then deallocates them when the let is exited.

Variables have their value set with the setq function which takes as its arguments a variable name and a value. So

```
(setq b 34.5)
```

sets the variable b to 34.5.

6.1.3 Program Control Structure

The if construct is a simple method for conditionally controlling the flow of a program; it is used in several places in the example programs. It takes a test clause, an expression to evaluate if the result of evaluating the test clause is true, and, optionally, an expression to evaluate if the result is false. The following simple example shows how if is used.

```
(if (= a 10)
  (print "a is 10")
  (print "a is not 10"))
```

Several of the examples use dotimes, a facility for executing a series of expressions a specified number of times. As an example,

```
(dotimes (j 10)
  (print j))
```

prints the integers from 0 to 9.

6.2 *Lisp Extensions

A *Lisp program looks much like an ordinary Lisp program. The biggest distinction is that *Lisp operations manipulate data stored in the Connection Machine hardware, while Lisp operates exclusively on the host processor. There are no instructions stored in the Connection Machine processors; instructions are generated from the *Lisp program and broadcast to the Connection Machine system.

The names of most *Lisp functions either begin with an “*” or end in “!!” (meant to look like two parallel lines, and pronounced *bang bang*) which means that they perform operations on parallel variables. This is only a naming convention and does nothing but distinguish functions that work with the Connection Machine system and parallel variables from functions that don't. User programs may also follow the convention, but it is not a requirement.

This section describes enough *Lisp to make the example programs understandable. As part of that, it is first necessary to describe a few of the fundamental features of the Connection Machine system.

6.2.1 Processors

A *processor* is the entity that operates on data in parallel. Each processor has a unique address that allows it to be directly accessed. The address is made up of one or more numbers depending how many dimensions the Connection Machine hardware is simulating. A one dimensional machine would take one number as an address, a two dimensional machine two numbers, etc. *Lisp has instructions that can directly access data in the Connection Machine processors via these addresses.

6.2.2 Parallel Variables

The parallel variable mechanism is one of the key programming differences between *Lisp and sequential programming languages. A thorough understanding of what parallel variables are and how they work is crucial to understanding the example *Lisp programs in this document.

On a serial machine a variable may have only one value at a time. On the Connection Machine system a parallel *Lisp variable has as many values as there are processors. Descriptors for parallel variables, or *pvars*, reside on the host computer, and the values of those parallel variables are in the Connection Machine memory.

The *Lisp expression for defining a pvar is similar to the Lisp mechanism for allocating a variable. The expression

```
(*defvar b (!! 5))
```

defines a pvar named b which has a value of 5 on every processor in the machine. The function `*defvar` is the parallel version of Lisp's `defvar`. The expression

```
(!! 5)
```

is the part of the `defvar` that actually does the allocation of a field with a value of 5 in every Connection Machine processor.

Values are retrieved from processors with the `pref` function. For example,

```
(pref b 7)
```

would return the value of pvar b in processor 7. Setting a value in a processor is accomplished with the Lisp `setf` function.

```
(setf (pref b 3) 10)
```

would set the value of pvar b to 10 in processor 3. The first argument to `setf` describes how to access the field that is going to be altered and the second argument is the new value of the field.

The following series of *Lisp expressions show in some detail how to allocate and use pvars.

First define some pvars:

```
(*defvar a)
```

```
(*defvar b (!! 5) "This is a documentation string.")
(*defvar c (!! -2.67))
(*defvar d t!!)
(*defvar e (1+!! (self-address!!)))
```

These statements created five pvars. The last four have been initialized with specific values: b is a Lisp symbol that has as a value a pvar whose contents is the integer 5 in each processor, c contains the floating point number −2.67 in each processor, d contains the boolean value true in each processor, and e contains the address of the next higher processor. The function self-address is a function that returns a pvar which contains the address of the selected processor.

Now read some of the values using pref.

```
(pref c 0)
```

returns the lisp value −2.67 since that is what is contained in pvar c in processor 0.

```
(pref d 365)
```

returns the lisp value t since that is what is contained in pvar d *in processor 0.*

Now do some arithmetic on these pvars:

```
(*set a (+!! b c))
```

will set the contents of pvar a to be the sum of the contents of pvar b and pvar c. Notice that c contains floating-point values. The integers contained in b are converted to floating-point numbers and the result in a will be floating point as well. Expressions can be nested:

```
(*set a (-!! b (*!! a (!! 2))))
```

This expression sets a to the difference of b and twice a. This simple expression could cause thousands of such operations to go on simultaneously! The expression (!! 2) returns a pvar that is 2 in all processors.

This point is important. The expression

```
(+!! a 2)
```

is an incorrect *Lisp expression. The variable a is a pvar, whose values are stored on the Connection Machine system, while the integer 2 is a Lisp object stored on the front end system. It is necessary to convert the 2 to a parallel value before doing any parallel computation.

6.2.3 Accessing Pvars Relative to a Grid

Two of the example programs, fluid flow and stereo matching, make heavy use of the Connection Machine system's grid mechanism, which facilitates communications between processors for problems with two-dimensional data structures. For example say `image` was a pvar containing a two-dimensional image. The following expression would shift the entire image over by one pixel in the x direction and place the result in `shifted-image`:

```
(*set shifted-image (pref-grid-relative!! image (!! 1) (!! 0)))
```

in this example the `(!! 1)` specifies that there is a shift of 1 in the x-dimension, and the `(!! 0)` specifies that there is no shift in the y-dimension.

6.2.4 Selection

In *Lisp it is possible to do an operation in a selected subset of all processors. The *Lisp function `*when` is used to do that selection. For example:

```
(*when (=!! a (!! 5))
  (*set a (+!! (!! 2))))
```

adds two to a in all processors in which a has a value of 5.

6.2.5 *Lisp Programs

*Lisp programs are defined in much the same way that Lisp functions are defined. The main difference is that `*defun` is used instead of `defun` to define functions that either take a parallel variable as an argument or return a parallel variable as a result.

6.3 Summary

*Lisp is a simple extension to Common Lisp that integrates the Connection Machine system into an ordinary serial programming environment. For someone familiar with Lisp, the essentials of *Lisp can be learned and put to productive use within a few hours.

Chapter 7

The Connection Machine System

The Connection Machine system from Thinking Machines Corporation is the first computer to implement data level parallelism in a general purpose way. It combines a very large number of processors with the communications capability necessary to match data topologies exactly. This chapter describes the hardware implementation of the Connection Machine system.

7.1 Connection Machine Internal Structure

As described in Chapter 1, the Connection Machine system operates by receiving a stream of instructions from its front end computer. A microcontroller receives the instructions, expands each of them into a series of machine instructions, then broadcasts the machine instructions, one at a time, to all processors at once. The instructions coming in from the front end are referred to as "macro-instructions." The instructions broadcast to the individual processors are called "nano-instructions." Macro-instructions are similar to assembly language instructions on a conventional machine. They are the instruction codes produced by the Connection Machine language processors. In the sections that follow, names of macro-instructions appear in italics.

The Connection Machine system includes 65,536 physical processors, but may be configured for a much larger number of logical processors by means of the *cold-boot* command. *Cold-boot* takes two arguments that allow a two-dimensional array of virtual processors per physical processor. *Cold-boot*(4, 4), for example, sets up the machine in the million-processor mode (or, more precisely, the 1,048,576 processor mode) because each of the 65,536 processors will simulate 16 (4×4) virtual processors. The same number of virtual processors could be established by the command *cold-boot* (16, 1). Since virtual processors are so commonly used, they are referred to simply as "processors". Where it is necessary to refer to one of the 65,536 hardware processors, the term "physical processor" is used.

Each physical processor has 4096 bits of memory, totalling 32 megabytes for the machine as a whole. In the million-processor mode, each processor has 256 bits of memory. Memory is divided into a data area and a stack area, with the layout being the same in each processor. A single, system-wide register, the stack limit, defines the boundary between stack space and data space. The stack pointer is also a system-wide register. The stacks in all processors act in unison.

Memory is bit-addressable; all data fields are of arbitrary length. For numeric computing there are three standard formats: unsigned-integer, signed-integer, and floating-point. Each is of arbitrary length. In particular, floating-point numbers can be of any length. Picture and word data are of arbitrary format and length.

A complete Connection Machine memory address has three parts. The first part indicates a physical processor. The second part indicates one of the virtual processors simulated by that physical processor. (This part is empty if there is only one virtual processor per physical processor.) The third part is an address within the memory of that virtual processor.

Data may be exchanged between the Connection Machine memory and the front end in any of three ways: slicewise, processorwise, and arraywise. *Read-slice* reads a single bit of information from the memory of each of a series of consecutive processors, assembles them into a signed integer, and passes the integer to the front end. *Write-slice* moves data from the front end to the Connection Machine memory. Slice operations are typically done 16 or 32 processors at a time. *Read-processor* and *write-processor* move a single field between the front end and a single processor. *Read-array* and *write-array* move arrays of fields between the front end and a set of contiguous processors.

7.2 Connection Machine Instruction Flow

All instructions flow into the Connection Machine hardware from the front end. These macro-instructions are sent to a microcontroller, which expands them into a series of nano-instructions. Some expand into just a few nano-instructions. Others expand into hundreds or thousands. It is also possible to feed nano-level instructions to the microcontroller and control the hardware directly. It is not, however, efficient to do so, because the front-end cannot supply these instructions rapidly enough to keep the system busy. (Direct control of the hardware from the front end is provided primarily so that the front end can support debugging and diagnostic aids.)

Nano-instructions are broadcast to all processors in parallel. Processors, however, have the option of "sitting out" a series of instructions. A one-bit flag within each processor, the *context flag*, determines whether that individual processor will respond to the instruction or not. Most of the instructions discussed in this chapter are "conditional" in the sense that they take effect only in the processors that are *active*, that is, whose *context flag* is 1.

The Connection Machine system is implemented with four physical microcontrollers, one for each section of 16,384 processors. If the system has a single front end, that front end is connected to all four microcontrollers and therefore drives all 65,536 processors. A system may be configured with up to four front ends. A crossbar switch called the Nexus makes the connections between front ends and microcontrollers. It is possible, therefore, to have four users operating simultaneously. Each works at a separate front end, and each has a separate instruction stream executing in a section of the system's processors. The examples in this chapter, however, assume that the system is operating with a single front end.

7.3 Computational and Global Instructions

Computational instructions operate on signed integers, unsigned integers, and floating-point values. They include unary operators such as ***not***, ***negate***, ***absolute value***, and ***square root***. All standard binary operators such as ***add***, ***subtract***, ***multiply***, ***divide***, ***compare***, and ***shift*** are included. These instructions operate in all processors simultaneously; each processor uses the data that is stored in that processor's memory.

The ***random*** instruction places an independently chosen pseudo-random number in each processor. Two processors may or may not be assigned the same random value.

Global instructions produce a single result from data items stored in the memories of all selected processors. ***Global-logior***, for example, takes the inclusive OR of a field in each processor's memory. ***Global-count*** examines a single-bit field in all processors and returns the number of "1" bits. ***Global-add*** sums multi-bit fields. ***Global-max*** and ***global-min*** return the largest (smallest) value found in a specified field across all selected processors. ***Global-add*** operates on unsigned integers, signed integers, or floating point values, as do ***global-max*** and ***global-min***. The ***enumerate*** instruction places a different consecutive integer into each of a selected set of processors.

7.4 Communications Instructions

The simplest form of communication between Connection Machine processors is between nearest neighbors. Each processor is wired to its neighbors to the North, East, West, and South by a communications network called the ***NEWS grid***. Four instructions, ***get-from-north***, ***get-from-east***, ***get-from-west***, and ***get-from-south*** control the transfer of data. Information is passed one bit at a time.

General intercommunication and dynamic reconfiguration is performed by a much more powerful communications system, the Connection Machine *router*. It allows full messages to be sent from any processor to any other; the sending processor simply needs to have the address of the destination processor. Messages may be of any length. Typical messages contain 32 bits of information; adding the address information and headers results in a transmitted package of 50 to 60 bits (depending on the number of virtual processors being used).

Each of the 65,536 physical processors is connected to 16 other physical processors in a special organization (a 16-dimensional hypercube) that provides large numbers of direct paths to distant parts of the system. Every processor is connected to 16 other processors, namely those whose binary address is different in just one of the 16 bits. The following example shows the interconnections of processors 6_{10} and 2070_{10}. The binary addresses are shown in parentheses.

```
    2 ( 0000 0000 0000 0010 )
    4 ( 0000 0000 0000 0100 )
    6 ( 0000 0000 0000 0110 )
    7 ( 0000 0000 0000 0111 )
   14 ( 0000 0000 0000 1110 )
   22 ( 0000 0000 0001 0110 )
   38 ( 0000 0000 0010 0110 )
   70 ( 0000 0000 0100 0110 )
  134 ( 0000 0000 1000 0110 )
  262 ( 0000 0001 0000 0110 )
  518 ( 0000 0010 0000 0110 )
 1030 ( 0000 0100 0000 0110 )
 2054 ( 0000 1000 0000 0110 )
 4102 ( 0001 0000 0000 0110 )
 8198 ( 0010 0000 0000 0110 )
16390 ( 0100 0000 0000 0110 )
32774 ( 1000 0000 0000 0110 )
```

```
   22 ( 0000 0000 0001 0110 )
 2054 ( 0000 1000 0000 0110 )
 2066 ( 0000 1000 0001 0010 )
 2068 ( 0000 1000 0001 0100 )
 2070 ( 0000 1000 0001 0110 )
 2071 ( 0000 1000 0001 0111 )
 2078 ( 0000 1000 0001 1110 )
 2102 ( 0000  1000  0011  0110 )
 2134 ( 0000  1000  0101  0110 )
 2198 ( 0000  1000  1001  0110 )
 2326 ( 0000  1001  0001  0110 )
 2582 ( 0000  1010  0001  0110 )
 3094 ( 0000  1100  0001  0110 )
 6166 ( 0001  1000  0001  0110 )
10262 ( 0010  1000  0001  0110 )
18454 ( 0100  1000  0001  0110 )
34838 ( 1000  1000  0001  0110 )
```

These two sets of addresses have a common connection. Processors 6 and 2070 both connect to 22. Thus it is possible to pass a message, for example, from processor 14 to processor 10262 in just four steps. The router at processor 14 passes it to the router at processor 6, which passes it to 22. From there it goes to 2070 and then to 10262.

7.5 The Routing Process

Connection Machine physical processors are grouped sixteen to a chip. There is a single router on each chip that services all sixteen processors. Hence four of the sixteen routing connections are internal to an individual chip. It takes a maximum of twelve steps to move from any chip to any other chip. During message routing, the system goes through all twelve steps. If the router on a given chip has a message whose relative address has a "1" in the low order bit position, it sends that message on the first of the twelve steps to the chip whose address differs in that same bit (i.e., the next chip). If the message it has has a "0" in the low order relative address bit, the on-chip router does not send any data on that step. The process continues through all twelve steps, with all router chips responding in the same way.

The basic message passing instruction is *send*. Arguments to *send* specify the length of the message and two memory fields. Within each processor, one field contains the message data, and the other contains the address of a destination processor. *Send* causes all active processors to initiate message transfers at once. The special Connection Machine routing hardware handles the volume of messages efficiently. An individual router on a chip may receive as many as twelve messages from other chips during a message cycle, one from each other chip that it is connected to. It can in turn send as many as twelve messages, one on each of the wires. If two messages need to go down the same wire, one is buffered until the next routing cycle. If an individual router becomes extremely busy, it can defer acceptance of any new messages from its own processors. Deferral keeps the router free to handle messages from other chips. If the chip's buffer space still fills, it refers messages to neighboring chips.

Simultaneous message sending introduces the possibility that the same location in the same processor will receive two or more messages in the same cycle. The simple *send* instruction gives unpredictable results in this case. Several variations of the *send* instruction, such as *send-with-add*, deal with this possibility. If two or more *send-with-add* messages arrive at the same destination, they are summed. *Send-with-overwrite* causes one message to be delivered intact, discarding all other messages directed to that destination. Other options include *send-with-max* and *send-with-logior*.

7.6 Dynamic Reconfiguration

A processor address is all it takes to establish a link on the system. This flexibility allows applications to reconfigure dynamically. A number of intructions support this capability. The *my-address* instruction allows processors to determine their own addresses, so they can send them to other processors and thus establish new connections. The *processor-cons* instruction allows each selected processor to find another "free" processor.

Processor-cons specifies the address of a one-bit field, the "free flag." A processor is considered free if it has a "1" in that field. The system looks in parallel for processors with 1's and passes to each selected processor the address of a different free processor, and at the same time clears the free flags of those free processors.

Chapter 8

Looking to the Future

At one level this report is about algorithms for data level parallel computers: algorithms for looking at the whole problem at once. But at a deeper and more important level, it is really the story of what happened when three very creative people teamed up with a new style of computer, the Connection Machine system. All three people saw new ways to break through old barriers. The computer allowed them to confirm their intuition quickly and then to build upon that intuition.

The intuitive insight behind the document retrieval algorithm is the fact that documents contain a rich set of synonyms for their main content topics. Comparing whole documents could eliminate the need to play guessing games with key words. The idea had never been effectively tested because no conventional computer could execute the algorithms quickly on large data bases. In fact, the first tests on document retrieval by whole document comparison were not particularly encouraging. They were run on a data base of 150 documents, which turned out to be inadequate. When the test was widened to 1500 documents, results were more encouraging. At the level of 15,000 documents, they were outstanding. Without a data level parallel computer such as the Connection Machine system, there would have been no way to even try the approach with 15,000 documents. Test runs would have taken days. Interaction would have been impossible. Now that it has been shown that the algorithm works, whole new possibilities for data base system design are opening up.

The intuitive insight behind the fluid flow algorithm is the fact the behavior of fluids can be simulated without extensive arithmetic computations. Modeling the primitive behavior of molecule packets on a large enough scale can elicit the same macroscopic behavior as real fluids. Tests on the Connection Machine computer suggest strongly that it does. The result is a new and potentially important avenue of scientific investigation.

The intuitive insight behind the contour mapping algorithm is the fact that sophisticated image processing and vision algorithms can be tested on large amounts of data with a small amount of programming effort. The drawing of contour maps, for example, is greatly simplified by data level parallelism, because it is not necessary to identify the contours one by one and then traverse the perimeter of each one sequentially; instead, each pixel of the contour map “draws itself” in parallel with all the other pixels. Instead of having to break up each phase of the problem into smaller pieces for sequencing purposes, the programmer can tackle it all at once. The result is smaller and simpler programs.

The revolution in data level parallel computing is here. The three algorithms described in this report are only a beginning. But they make an important point: innovative users are an integral part of the story. Users who are stimulated to look at old problems in new ways. Users who revisit problems given up on as impossible in the 60's and 70's. Users who know that a simpler solution is a better solution. These are the users who will assure that the future belongs to computers that look at the whole problem at once.

Bibliography

[1] David C. Blair and M. E. Maron. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Comm. ACM*, 28(3):289–267, March 1985.

[2] John F. Canny. *Finding Lines and Edges in Images.* AI Memo 720, MIT Artificial Intelligence Laboratory, Cambridge, Massachusetts, 1983.

[3] Michael Drumheller and Tomaso Poggio. On parallel stereo. In *International Conference on Robotics and Automation*, IEEE, April 1986.

[4] U. Frisch, B. Hasslacher, and Y. Pomeau. *A Lattice Gas Automaton for the Navier-Stokes equation.* Preprint LA-UR-85-3503, Los Alamos, 1985.

[5] W. Eric L. Grimson. *From Images to Surface.* MIT Press, Cambridge, Massachusetts, 1981.

[6] Samuel P. Harbison and Guy L. Steele Jr. *C: A Reference Manual.* Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[7] J. Hardy, O. de Pazzis, and Y. Pomeau. Molecular dynamics of a classical lattice gas: transport properties and time correlation functions. *Phys. Rev.*, A13(1949), 1976.

[8] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, Massachusetts, 1985.

[9] Guy L. Steele Jr., Scott E. Fahlman, Richard P. Gabriel, David A. Moon, and Daniel L. Weinreb. *Common Lisp: The Language.* Digital Press, Burlington, Massachusetts, 1984.

[10] Brian W. Kernighan and Dennis Ritchie. *The C Programming Language.* Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[11] David Marr. *Vision.* W. H. Freeman, San Francisco, 1982.

[12] David Marr and Ellen Hildreth. Theory of edge detection. *Proc. Roy. Soc. London*, B(207):187–217, 1980.

[13] K. Prazdny. Detection of binocular disparities. *Biological Cybernetics*, 52:93–99, 1985.

[14] James B. Salem and Stephen Wolfram. *Thermodynamics and Hydrodynamics with Cellular Automata*. Internal technical report, Thinking Machines Corporation, Cambridge, Massachusetts, Noνember 1985.

[15] David S. Touretzky. *Lisp: A Gentle Introduction to Symbolic Computation*. Harper & Row, New York, 1984.

[16] Patrick Henry Winston and Berthold Klaus Paul Horn. *Lisp*. Addison-Wesley, Reading, Massachusetts, second edition, 1984.